LiveShift: A Time-Shifted P2P Multimedia Streaming Approach

Doctoral Thesis

For the Degree of
Doctor of Informatics

At the Faculty of Economics,
Business Administration And
Information Technology
of the
University of Zurich

by
Fabio Victora Hecht
from
Brazil

Accepted on the recommendation of
Prof. Dr. Burkhard Stiller
Dr. Tobias Hossfeld

2013

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, January 23, 2013

The Vice Dean of the Academic Program in Informatics: Prof. Dr. Harald C. Gall

Abstract

Interest in multimedia streaming – download and immediate playback of audio and video content – on the Internet has grown immensely in the last decade. Traditional client/server (C/S) and content distribution network (CDN) architectures, though technically simpler, are expensive, since they require infrastructure to grow according to the number of simultaneous clients. This results in the existence of only a few large content providers, thus support for transmission of events without great commercial appeal is restricted. The peer-to-peer (P2P) architecture, on the contrary, allows low-cost and scalable distribution of multimedia content by users themselves, without requiring large infrastructure investments.

Multimedia streaming is essentially classified into three categories, concerning its distribution and consumption. Live streaming targets live events, such as sports, news, and ceremonies, with the following important technical characteristics: no defined start or ending, generation on-the-fly, limited buffering ability, quick expiration of content, and lack of video-cassette recorder (VCR) functions, e.g., pausing and rewinding. Video-on-Demand (VoD), commonly used for pre-recorded content, including movies and shows, assumes that users start playback from the beginning of the file, content has a known duration, maximum download rate is not limited by content generation rate, and sometimes allows VCR operations, depending on the particular implementation. Finally, time shifting (TS) is a relatively newer category that bridges the gap between live streaming and VoD by enabling users to conveniently record live streams and watch them at a future time, at their convenience, thus enabling VCR functions on the live stream. Though P2P-based live streaming and VoD systems have gained significant attention of the research community, with commercial implementations already released to the general public, a single P2P system capable of both live streaming and time shifting did not exist.

The seamless integration of live, TS, and VoD in a single system originates several new challenges. First, the ability of users to switch both channels and time positions and the asymmetry of interest inherent in such sce-

nario require the definition of a novel, suitable, unified protocol, as well as the investigation and definition of policies that drive its behavior. A rather understudied policy is the playback policy, applied when content is not downloaded quickly enough to be played – shall peers stall playback, waiting for content to be found and downloaded, or skip missing parts? Finally, P2P system architectures demand distributed components, e.g. P2P trackers, that improve both efficiency and load balancing.

Hence, this thesis investigates several aspects that make the combination and seamless unification of P2P live streaming and time shifting functionality under a single system, protocol, and policies possible. This is accomplished by building a system – called LiveShift – that allows peers to record parts of the stream they receive and in order to serve them to other peers at a future time when necessary. In practice, this allows users to conveniently watch any program at any time, without preparing recordings in advance, and scaling bandwidth and storage with the number of users in the system.

This thesis makes four main contributions in the field of time-shifted P2P multimedia streaming. First, a single, fully-distributed P2P protocol that is capable of locating and distributing both live and time-shifted streams quickly and in an integrated manner, is described and evaluated in detail. Second, a complete client implementation of LiveShift was built, in order to allow realistic experiments and demonstrations. Third, playback policies that define whether the system stalls playback or skips blocks when content is unavailable in the system are examined, compared, and classified regarding user experience, for both live and on-demand scenarios. Finally, a novel, fully-distributed P2P tracker – named B-Tracker – is introduced, improving efficiency and load balancing in comparison with other existing distributed P2P trackers.

LiveShift has been evaluated by means of a full software implementation with the ultimate goal of validating its protocol and default policies in a realistic environment, by examining objective Quality-of-Experience metrics, namely playback lag and share of skipped blocks. Evaluations include realistic heterogeneous P2P environments with channel switching and churn in a research testbed. Moreover, a wide range of playback policies was implemented to show how they effect the defined Quality-of-Experience metrics. Finally, experiments with B-Tracker have revealed its ability to improve both efficiency and load balancing in relation to other widely used P2P trackers.

Kurzfassung

Das Interesse an Multimedia-Übertragungen – zum Herunterladenwie auch zur unmittelbaren Wiedergabe von Audio- und Video-Inhalten – im Internet ist im Laufe des letzten Jahrzehnts stark gewachsen. Traditionelle Client/Server (C/S) und Content Distribution Network (CDN) Architekturen sind teuer, auch wenn sie technisch einfacher sind, weil ihre IT-Infrastruktur mit der Anzahl gleichzeitig zu bedienender Clients mitwachsen muss. Aus diesem Grund gibt es nur einige wenige grosse Inhalteanbieter, was wiederum dazu führt, dass die Möglichkeiten zur Übertragung von Veranstaltungen ohne grosses kommerzielles Potential stark eingeschränkt sind. Die Peer-to-Peer (P2P) Architektur ermöglicht hingegen die kostengünstige und skalierbare Verbreitung von Multimedia-Inhalten durch die Nutzer selbst und ohne grosse Infrastrukturinvestitionen.

Multimedia-Übertragungen werden anhand der beiden Kriterien Verbreitung und Konsum im Wesentlichen in drei Kategorien unterteilt. Direktübertragungen zielen auf Anlässe von aktuellem Belang ab, zum Beispiel Sportveranstaltungen, Nachrichten und Feierlichkeiten. Sie zeigen die wichtigen technischen Eigenschaften eines nicht vorbestimmten Anfangs oder Endes, der unterbrechungsfreien Generierung, begrenzter Möglichkeiten zur Pufferung, des schnellen Veraltens der Inhalte und der Abwesenheit von gängigen Videorekorder-Funktionen, beispielsweise dem Pausieren und Zurückspulen. Video auf Abruf (Video-on-Demand, VoD), welches als zweite Kategorie üblicherweise Anwendung auf bereits aufgenommene Inhalte findet (einschliesslich Filme und Fernsehsendungen) basiert auf der Annahme, dass Nutzer die Wiedergabe am Anfang einer Datei starten, dass die Inhalte eine vorgängig bekannte Dauer haben, dass die maximale Download-Rate nicht von der Rate begrenzt wird, mit der Inhalte generiert wurden, und dass in Abhängigkeit von der spezifischen Umsetzung zuweilen Videorekorder-Funktionen unterstützt werden. Schliesslich bezeichnet die zeitversetzte Übertragung (Time Shift-

ing, TS) die dritte und im Vergleich jüngere Kategorie, die die Lücke zwischen Direktübertragungen und VoD füllt, indem sie es Nutzern erlauben bequem Direktübertragungen aufzunehmen, um sie zu einem späteren und passenderen Zeitpunkt wiederzugeben, womit zeitversetzte Übertragungen Videorekorder-Funktionen bei Direktübertragungen ermöglichen. Obwohl P2P-basierte Direktübertragungs- und VoD-Systeme den Gegenstand von beträchtlichen Forschungsvorhaben darstellen, wobei einige kommerzielle Lösungen bereits der Öffentlichkeit zugänglich gemacht wurden, gab es zu Beginn der vorliegenden Dissertation kein einziges P2P-System, das sowohl für Direktübertragungen als auch für TS geeignet gewesen wäre.

Die nahtlose Integration von Direktübertragungen, TS und VoD in einem einzigen System resultiert in mehreren neuen Herausforderungen. Dass Nutzer sowohl Kanäle wie auch Wiedergabe-Zeitpunkte wechseln können und die einem solchen Szenario inhärente Informationsasymmetrie bedingen die Definition eines neuen, geeigneten und einheitlichen Protokolls sowie die Untersuchung und Definition von Regeln, die erwünschtes Verhalten fördern. Ein bis anhin kaum erforschter Bereich stellt dabei ein Satz an Regeln für die Wiedergabe von Inhalten dar, die sogenannte Playback Policy, die Peer-Verhalten definiert für den Fall, dass Inhalte nicht schnell genug heruntergeladen werden, um problemlos abgespielt zu werden – sollen Peers in dieser Situation die Wiedergabe pausieren und warten, bis die fehlenden Inhalte gefunden und heruntergeladen wurden, oder ist es besser, fehlende Teile zu überspringen? Und schliesslich bedingen P2P Systemarchitekturen verteilte Komponenten, zum Beispiel P2P-Tracker, die für Verbesserungen in Bezug auf Effizienz und Lastverteilung sorgen.

Daher untersucht diese Dissertation mehrere Aspekte, die die Kombination und nahtlose Vereinigung von direkter und zeitversetzter Übertragung in einem einzigen P2P-System unter Verwendung eines einheitlichen Protokolls und einheitlicher Regeln ermöglichen. Dies wird durch die Umsetzung eines neuen Systems erreicht – LiveShift genannt –, das es Peers erlaubt, Teile einer empfangenen Übertragung aufzuzeichnen, um diese Aufzeichnungen zu einem späteren Zeitpunkt bei Bedarf anderen Peers anbieten zu können. In der Anwendung bedeutet dies für die Nutzer, dass sie jederzeit bequem auf alle Programme zugreifen können, ohne dass sie

im Voraus Aufnahmen programmieren müssen, wobei Bandbreite und Speicherplatz mit der Anzahl der Nutzer im System skalieren.

Die Dissertation leistet damit vier zentrale Beiträge im Bereich zeitversetzter P2P-basierter Multimedia-Übertragungen. Erstens beschreibt und evaluiert sie detailliert ein einziges, vollständig verteiltes P2P-Protokoll, das gleichermassen direkte wie zeitversetzte Übertragungen schnell lokalisieren und verteilen kann. Zweitens erfolgte die komplette Umsetzung einer LiveShift-Anwendung für Nutzer des Systems, was realitätsnahe Experimente und Vorführungen ermöglichte. Drittens untersucht, vergleicht und klassifiziert die Dissertation Playback Policies, die festlegen, ob das System die Wiedergabe anhält oder ob es Teile überspringt, wenn Inhalte nicht verfügbar sind, auf ihre Auswirkungen auf das Nutzererlebnis für die beiden Szenarien einer Direktübertragung und einer Übertragung auf Abruf. Schliesslich führt sie einen neuartigen, voll verteilten P2P-Tracker – benannt B-Tracker – ein, der Effizienz und Lastverteilung im Vergleich zu anderen existierenden verteilten P2P-Trackern verbessert.

LiveShift wurde mittels einer vollständigen Umsetzung der Software evaluiert zwecks Validierung seines Protokolls und der Standard-Policies in einer realitätsnahen Umgebung unter Verwendung objektiver Metriken zur Messung des Nutzererlebnisses, nämlich der Wartezeit bis zum Start der Wiedergabe (Playback Lag) und dem Anteil übersprungener Blöcke. Die Evaluation wurde in einer Forschungs-Testumgebung durchgeführt und spiegelt realitätsnahe heterogene P2P-Umgebungen wider mit entsprechend modellierten Kanalwechseln und Churn (Peers, die das System verlassen). Darüber hinaus wurde ein breites Spektrum von Playback Policies umgesetzt, um ihren Einfluss auf die betrachteten Metriken zur Messung des Nutzerelebnisses zu zeigen. Schliesslich bestätigten Experimente mit B-Tracker die angenommene Effizienz-Steigerung und verbesserte Lastverteilung gegenüber anderer verbreitet eingesetzter P2P-Tracker.

Contents

1 Introduction

Technology advances of the last decades, in both hardware and software, have allowed computer networks and distributed systems to revolutionize society by enabling novel communication possibilities [112]. Such revolution stems from the fact that sharing information among numerous users on the Internet has become more efficient and convenient. That includes, for instance, electronic mail, blogging, and social networks.

1.1 Multimedia Content Distribution

An important form of communication is the transmission of multimedia content – e.g. audio, video – to large audiences, historically having been performed using technologies such as radio, television, Video Home System (VHS), and DVD. The use of computer networks for this purpose offers several advantages. First, it allows a much larger content variety, since any user may become a content provider. Besides, it enables content browsing and reception from locations which are convenient for users – their living room or mobile devices. Also, the integration with other distributed systems, providing, for instance, metadata and movie trailers, or social networks that provide recommendations, multiply the possibilities of enter-

tainment. Consequently, the demand for multimedia streaming on the Internet is increasing [53]; distributed systems such as YouTube [123], SopCast [107], and BitTorrent [9] already attract a large audience for exchange of multimedia content.

However, the transmission of multimedia content to large audiences via computer networks is inherently challenging. First, such type of content has large bandwidth requirements and is highly delay-sensitive [53]. Second, the shared media present in today's networks, while in one hand allowed them to grow at a relatively lower cost, has resulted in lack of bandwidth and delay guarantees. This makes it expensive, and therefore unfeasible, for end users and small producers to broadcast content without using expensive distributed systems provided by large enterprises.

The high cost is a result of the client/server (C/S) architecture adopted by such distributed systems. Content is stored at servers that establish a one-to-one (unicast) connection to each client, at which users are located. Such centralized architecture is technologically simpler than distributed ones, thus are easier to develop, debug, and control; however, it suffers from cost and scalability issues, since the number of servers and their upstream bandwidth must scale with the number of users present. A large, geographically-distributed C/S system – named content distribution network (CDN) [100] – achieves high-performance, reliable transmission of content to a large audience; it is, however, very costly to build and maintain [91].

The peer-to-peer (P2P) architecture, an alternative to C/S, has shown to be effective both to increase scalability and to decrease media publishing costs, as more users join the multimedia system. This is accomplished by allowing users to collaborate by sharing (often unused) resources, such as upstream bandwidth and disk storage, with the broadcaster, reducing its costs and increasing scalability. The performance of a P2P architecture has been recently increasing, due to the growth in deployment of Fiber to the Home (FTTH) technology [106], which increases upload capacity of end users, and the escalation in IPv6 deployment [64], which makes them better reachable. This, combined with advances in P2P protocols and algorithms, results in P2P quickly becoming a mature technology, with

wide-scale deployments already providing audio and video distribution services [9, 95, 107]. Nonetheless, the distributed nature of P2P systems and the lack of central control increase its overall complexity; P2P is, hence, currently an active field of research.

Though multimedia streaming is fundamentally classified, according to how streams are distributed and consumed, into two categories – live streaming and Video-on-Demand (VoD) [77] –, this thesis is concerned with a relatively newer category – time shifting (TS) [48]. *Live streaming* is similar to radio and television transmissions, in the sense that each broadcaster continually transmits the stream to a set of users, who may only receive the newest data. Video Cassette Recorder (VCR) operations, namely pause, fast-forward and rewind are, thus, not available. This is due to live streaming being technologically the simplest case of all, since only a few seconds need to be stored for buffering purposes. SopCast [107] is a popular P2P application that provides live streaming. *VoD*, on the other hand, refers to streaming of content (e.g., a movie) at the time it is requested by the user. Since this normally assumes some storage to buffer content at the client side and data is required to be transmitted also from the receiver to the sender (for selecting and starting content playback), VCR operations can be implemented. Tribler [95] is a popular P2P application that provides VoD. Finally, *time shifting* is a relatively newer concept, provided by a digital video recorder (DVR), such as TiVo [114], that combines the advantages of live streaming and VoD, by letting users utilize the live stream for real-time content, such as sports events, breaking news, and movie premieres, while allowing them to conveniently record the live stream. This allows the use of VCR operations on the live stream to, like in VoD, enable pausing and watching the content at a later, more suitable occasion.

Time-shifting is showing a dramatic increase in popularity; for instance, in the USA, as of 2010, more than one-third of all TV owners own a DVR, up 51 percent from the year before, and time shifting watching time has been up 14.7 percent in the same period, while high-speed broadband Internet access was in 63.5 percent of homes [24, 25]. Figure 1.1 shows a Swiss street advertisement from February 2012 introducing the new time-shifting functionality – commercially called "ComeBack TV" – to the pub-

lic; time shifting, though, is limited to 40 channels and the last 28 hours, since the systems follows a C/S architecture. At the time of the start of this thesis work (mid 2008), no fully-distributed P2P system supporting live streaming and time shifting existed [48].

Hence, this thesis explores the opportunity of allowing collaboration beyond the current state of the art by using a single P2P architecture [85] to deliver both live and time-shifted multimedia streams. This enables users to store received live streams in order to distribute them in the future, thus allowing time shifting (TS) or – if the combined storage is large – also VoD functionality.

Figure 1.2 pictures an example of the novel use case targeted. While some peers are interested in receiving the live stream, others wish to watch it at a later time. This is made possible because streams are selectively recorded by every peer, such that they can be transmitted at a later time, as convenient. Each peer runs the same application, called *LiveShift*. The collective of peers, applications, and their interactions is referred to as the *LiveShift system*. This use case allows a user, without having previously prepared any local recording, to watch a program from the start and jump over uninteresting parts until seamlessly catching up with the live stream. Due to this thesis contributions, watching pre-recorded or live content does not require different protocols and system architectures; these two functions are seamlessly integrated into a single and novel protocol. Similarly, live transmission may be used for the premiere of a movie, TV show, or news program, when several users might watch it at the same time. Instantly and automatically, the content is made available – since the starting time of the premiere – to every user joining at a later time. Content providers may also benefit from the proposed approach, since, besides general P2P properties that reduce bandwidth costs at the provider side, moving the storage to the end-user results in content automatically being replicated at a factor proportional to its popularity and distributed to locations where it is popular. This increases both content reliability and availability. The novel protocol does not mandate particular architectural requirements, as it can be deployed on computers, set top boxes, or servers provided by Internet service providers

Figure 1.1: Billboard advertisement of a Swiss TV operator that allows time shifting, February 2012. In German. "Missed movie? No problem. With ComeBack TV you can watch programs up to 28 hours later, because we record the entire program from 40 channels automatically for you."

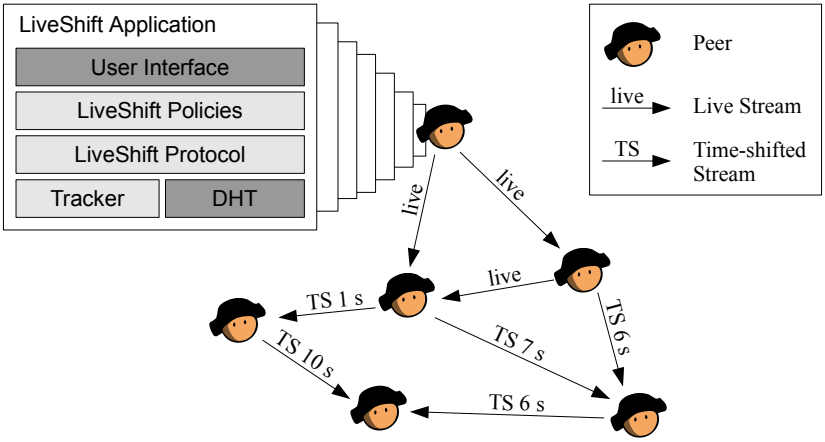

Figure 1.2: LiveShift system and use case example. Lighter boxes highlight aspects targeted in this thesis.

(ISPs) or content distribution network (CDN) operators; providers may include large broadcasters, as well as smaller home-based ones.

The LiveShift application is made up of several elements, but three research dimensions, also pictured in Figure 1.2, are targeted particularly in this thesis. These they differ significantly from current P2P streaming applications, which support either live or on-demand streaming, and are introduced in the following subsections. The first is the *LiveShift protocol*, a flexible protocol that allows peers to exchange both live and time-shifted content in a fully-distributed manner. The possibility of uniting such functionality under the same protocol and application requires the analysis and comparison of *playback policies* under a variety of scenarios, ranging from under- to over-provisioned in total upload bandwidth available, in order to conclude which are best suited for which situation. Finally, since the architecture is fully-distributed, the development of *B-Tracker*, a P2P *tracker* with improved load balancing and efficiency properties is as well required.

1.2 Basic Terminology

The following terminology is used in this thesis.

Stream: Continuous data transmission with no obvious start and end, *c.f.* Section 1.1.

Multimedia content: Combination audio, video, and possibly other (e.g. text) streams, *c.f.* Section 1.1.

Live streaming: Continuous transmission and immediate consumption of multimedia streams, *c.f.* Section 1.1.

Video-on-Demand (VoD): Streaming at the time it is requested by the user, *c.f.* Section 1.1.

Time shifting (TS): Seamless switching between live streaming and VoD, *c.f.* Section 1.1.

Client/Server (C/S): Architecture in which participants are split into providers (servers) and consumers (clients), *c.f.* Sections 1.1, 2.1.

Peer-to-peer (P2P): Every participant has both provider and consumer abilities, *c.f.* Sections 1.1, 2.1, 2.2, 2.3.

Quality-of-Service (QoS): Quality measurement of a service typically based on packet-level metrics, *c.f.* Section 2.4.

Quality-of-Experience (QoE): Subjective and objective metrics for user-centric quality measurement, *c.f.* Section 2.4.

1.3 Integrated Live and Time-Shifting P2P Streaming Protocol and Application

Integrating live, time-shifted, and on-demand multimedia streaming in a single system introduces several new challenges, since research has thus far treated them completely separately, as described in Section 2.3.3; however, three challenges are distinguished. First, and in stark difference with a live streaming system, users may switch not only between channels, but also within various time positions in a given channel, over a potentially large time scale. Second, the asymmetry of interest [14] inherent in such scenario demands a flexible protocol that does not require that peers be simultaneously interested in data each other has. Third, the presence of a unified protocol that supports both live and on-demand characteristics requires the definition of policies, that is, specific algorithms that define peer behavior in relationship to the protocol, that are effective on all scenarios.

This thesis identifies the decoupling between protocol and policies. While the protocol definition characterizes the syntax and semantic of messages exchanged between peers, local decisions, such as when to send such messages to which peers, how to prioritize allocation of scarce resources (e.g., bandwidth and storage), and how to behave when requests time out or content is unavailable, are defined as policies. The protocol is flexible regarding the definition of different policies, which is a recognition of the fact that peers will attempt to define policies according to their own preference and interest.

Therefore, in this thesis, the *LiveShift protocol* [46], a new, flexible protocol that supports the envisioned use case, and a set of preliminary policies to be used with the protocol, are defined and evaluated. Protocol evaluation uses a full reference implementation – called the *LiveShift application* – , to allow user experience to be realistically evaluated, as well as interdependencies among different policies to be investigated, a full reference client implementation was also developed. While simulations would allow evaluating a larger number of peers in the system, these produce less realistic results, since a simplified logic is employed and many assumptions are in place.

The LiveShift protocol and a set of initial policies have been researched, designed and implemented as part of the LiveShift application in order to be evaluated under realistic assumptions. User behavior has been defined from traces [18], including channel switching patterns. Three realistic scenarios, with different levels of scenario load, have been selected, to show system performance in a variety of situations that are increasingly challenging for P2P systems. Playback lag, described in Sections 3.5 and 4.1, is employed as Quality-of-Experience (QoE) indicator metric – also referred to, for the sake of readability, as *QoE metric*. Evaluation results show that the defined use case of integrating live and time-shifted multimedia streaming is supported in a fully-distributed scenario. Playback lag remains lower than 60 seconds of transmission for over 95 percent of the peers in the investigated scenarios. The system also shows good resilience to churn (peers joining and leaving the system), and switching channels. The LiveShift protocol is the first fully-decentralized, mesh-pull protocol designed for both

live streaming and VoD, and which has been tested to include the effects of peer channel browsing behavior.

1.4 Playback Policies

Content availability in LiveShift or any other streaming system is affected by network and server conditions, possibly causing content not to be downloaded on time to be played. The challenge increases in P2P systems due to, e.g., poorly managed networks, asymmetric bandwidth of peers, traffic-shaping at ISPs, free-riding, limited view of peers, and the fact that users change their interest frequently – switching channels and, in case of LiveShift, also time shifting. Content may even be available in the P2P system, but not downloaded before its playback, due to, for example, the only peers that can provide the respective content having allocated all their upload capacity to serve other peers. The term *content availability* is, thus, defined in terms of content downloaded before its playback deadline.

This thesis defines the *playback policy* as the decision on, when content is unavailable, whether to stall playback, or skip to a block that has already been downloaded. Though any P2P multimedia streaming system needs to implement a playback policy, current systems either omit this information, or adopt an arbitrarily-defined policy. Out of the set of policies identified in the development of the LiveShift protocol, the playback policy is the one that has received the least attention by the research community. Thus, this thesis also defines, compares and documents the effect of different playback policies under both live and on-demand situations.

In the scope of playback policies, this thesis investigates how different playback policies affect user experience in a P2P multimedia streaming system, and defines which playback policies are most suitable for live and on-demand scenarios [49]. A classification and generalization of different playback policies are therefore performed, allowing them to be fairly compared under a constant set of parameters. These policies have been implemented in LiveShift, in order to be evaluated using a real client implementation, and compared under a variety of carefully-selected scenarios and parameters, as follows. Scenarios ranged from under- to over-loaded

regarding the ratio between total number of peers and total upload capacity. Policy parameters were selected based on values seen in the literature, complemented by additional numbers that reveal interesting properties.

1.5 Fully-distributed P2P Tracker

Having defined the LiveShift protocol with the desired properties and investigated playback policies under live and on-demand streaming scenario, this thesis has identified the opportunity of advancing the state-of-the-art on fully-distributed P2P trackers. Trackers [9] are important building blocks used in P2P systems for provider discovery – mapping *resources*, such as files or stream segments, to *providers*, that is, peers that announce the ability to provide them.

Today's most popular P2P systems, e.g., the original BitTorrent [23], employ a hybrid architecture, in which although large data transfers are done using a P2P approach, the tracker still follows a C/S approach. C/S-based trackers do not fully benefit from P2P properties, such as no single point of failure, scalability, load balancing, and the lack of a central authority. Therefore, different types of distributed trackers have been deployed. Distributed hash tables (DHTs) are natural candidates and thus have been used as distributed trackers [26, 109], since their main functionality is mapping keys (content) into values (providers), but these show poor load balancing characteristics, since content popularity distribution is approximately power law. Another way of designing a distributed tracker is using a gossip protocol, such as Peer Exchange (PEX) [10, 92], to allow peers to spread information about potential providers, but information is spread is slowly and inefficiently.

This thesis, thus, defines a novel distributed tracker, named B-Tracker (Balanced Tracker) [50], to improve both efficiency and load balancing of fully-distributed P2P trackers. The main idea is that each provider becomes itself a tracker for the resources it provides, and Bloom filters [11] are used to avoid unnecessary redundancy. B-Tracker may be used not only by LiveShift, but by any P2P application that uses a tracker to locate possible providers. These include file-sharing applications, such as BitTorrent,

but delay-sensitive applications, such as LiveShift, enjoy additional benefits by requesting peers when those are needed, since B-Tracker is pull-based. Evaluations show that the proposed approach achieves both better efficiency and load balancing when compared to a pure DHT and PEX approaches.

1.6 Thesis Contributions

Motivated by the above observations, this thesis makes the following contributions to the field of time-shifted P2P multimedia streaming:

1. developing, implementing, and evaluating a flexible and fully-decentralized P2P *protocol*, capable of supporting live streaming, time-shifting, and VoD in an integrated manner, while taking asymmetry of interest into account, being efficient, tolerating peers joining and leaving the system and switching channels, allowing different policies to be instantiated, ultimately showing the system's ability to find content and upstream capacity quickly enough to sustain a low playback lag relatively to the playback position;

2. defining the term *playback policy*, as well as defining and investigating playback policies under different scenarios and parameters, to understand how they affect user experience on P2P multimedia streaming systems, and finally, which playback policies are most suitable for each situation;

3. introducing, defining, and evaluating a pull-based, *fully-distributed tracker* capable of finding content providers with improved efficiency and load balancing when compared to the current state-of-the-art; and

4. developing a reference *client implementation* for overall instantiation and validation of the main concepts of this thesis in a practical manner, allowing presentation of demonstrations, and contributing to the open-source and research communities.

This thesis is based on several publications that compose contributions in the following areas: LiveShift protocol, policies, and client implementation [45, 46, 48]; playback policies [49]; and B-Tracker [50, 51]. These contributions do represent important and innovative advancements in both research and operations of P2P multimedia streaming systems.

1.7 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related work on concepts that lay the technical foundation upon which this thesis stands; that includes file-sharing and multimedia streaming systems in general, including on systems that provide time-shifting functionality, as well as more specifically playback policies and distributed trackers.

Chapter 3 defines the LiveShift architecture, protocol, and policies, including implementation and evaluation results. Evaluation was performed in a test bed, with the system fully implemented, by relying on realistic peer behavior modeled from traces, including channel switching and churn, to successfully capture the defined QoE metric of a real running system. This validates the LiveShift Protocol and initial set of policies in a realistic environment, setting the foundation for supporting the envisioned use case.

Out of the important policies that have been identified, Chapter 4 analyzes playback policies in depth, defining and evaluating with experiments the performance of different playback policies for live and on-demand video streaming under a wide range of parameter and scenario choices. Evaluations were also made in a test bed using trace-driven peer behavior, focusing on the QoE metric obtained with different playback policies and parameters.

Since the system is fully-distributed and must show good efficiency and load balancing properties, Chapter 5 focuses on the problem of distributed trackers, more specifically the B-Tracker approach, introducing its definitions and evaluation results that show its better efficiency and load balancing properties compared to other distributed tracker approaches. Experiments focus on efficiency and load balancing of tracker messages and to

stress that B-Tracker can be used for a variety of P2P applications, including both streaming and file sharing.

LiveShift has been fully implemented and published as an open-source application [78]. Chapter 6 features the LiveShift application, including screenshots that illustrate its main functionality. In addition, implementation of playback policies and B-Tracker in LiveShift are specified.

Finally, Chapter 7 concludes this thesis, summarizing contributions and key findings, and suggests future work.

2
Related Work

Using technology for sharing information, including multimedia content, has been done for decades, using, for instance, devices for copying, storing, distributing, and reproducing music and video, such as cassette and VHS tapes. The development of computer networks and distributed systems, especially P2P systems, has transformed, though, the way users collaborate and share content [85], by allowing massive content sharing. This chapter presents technologies which are important building blocks for LiveShift, including P2P file sharing systems, P2P multimedia streaming systems, Quality-of-Service and Quality-of-Experience, playback policies for video streaming, and distributed tracker approaches. Since there are many commercial systems that do not reveal their specific algorithms (e.g. SopCast [107]), their details are not made available.

2.1 Content Distribution Architectures

In a C/S architecture, each client that is interested in receiving data (e.g., downloading a file or a video stream) establishes a connection to a central server, as illustrated in Figure 2.1. This means that the server must potentially store and upload a very large amount of data, which creates a

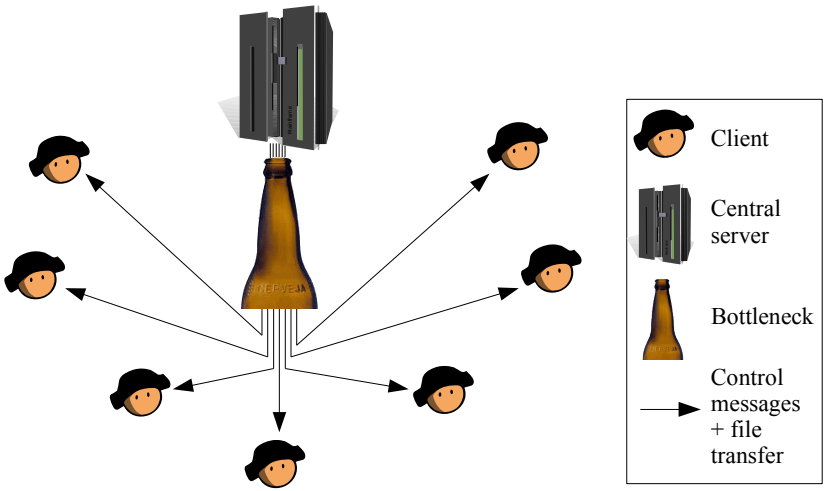

Figure 2.1: C/S architecture

bottleneck at the server's network connection and storage devices. Thus, the number of servers, their bandwidth, and hence, their maintenance cost, grow as the number of clients increases [85].

P2P systems have emerged in the late 1990s as distributed systems that use computer networks, in particular the Internet, to allow users to explore often unused resources, such as bandwidth, central processing unit (CPU) cycles, and storage, to collaborate on providing a service – for example, transferring files or streaming multimedia content. Such systems eliminate the bottleneck at the server by reducing their load significantly (in case of hybrid architectures), or even (in case of pure-P2P architectures) eliminating completely the need for servers, since the server role is distributed through the clients. The software running at the users' premises are called peers [85], since they are functionally identical – all peers are able to perform both server and client roles. SETI@home [104] is an example of an early P2P system – released in 1999 – that utilizes user CPU power in the search for extraterrestrial intelligence (SETI) by analyzing radio telescope data.

In order to provide the intended services, P2P systems often build an *overlay* network [21], on top of the Internet, that provides independent addressing and routing schemes. This gives P2P systems the freedom to implement their own addressing and routing protocols and algorithms, independently from the underlying network.

Much of the technology used for P2P multimedia streaming traces its origins to P2P file sharing. Thus, an overview on P2P file sharing systems and protocols is first presented in this chapter, providing essential background, historical, and technical information. Then, existing approaches for P2P multimedia streaming, including live streaming, VoD, and finally time-shifting systems, are described, the latter being compared regarding important properties, such as application in a fully-distributed architecture, overlay topology, and evaluation of user experience. Then, a survey of playback policies adopted by existing P2P streaming systems is presented. Finally, related work about P2P trackers, important elements in P2P systems for peer and content discovery, is described and compared.

2.2 P2P File Sharing Systems and Protocols

It was in June 1999 that the P2P system Napster was released, offering a file sharing service that was used primarily for distribution of multimedia content, in particular music [101]. Napster employed a hybrid architecture, as illustrated in Figure 2.2, in which file transfers happened directly between pairs of peers, decreasing server traffic considerably. A central server was still in place, being responsible for performing keyword search and keeping mappings between files and peers that held them. A peer interested in downloading a file would first contact the server to retrieve a list of peers providing the file; then, each of these peers directly to set up the file transfer. Napster's P2P architecture has revolutionized file sharing, and its usage has peaked with more than 26 million users worldwide in February 2001 [38]. However, due to having been repeatedly sued for "engaging in, or facilitating others in, copying, downloading, uploading, transmitting, or distributing copyrighted musical compositions or sound recordings" [16], Napster was forced to shut down its entire network in July 2001. Shutting down

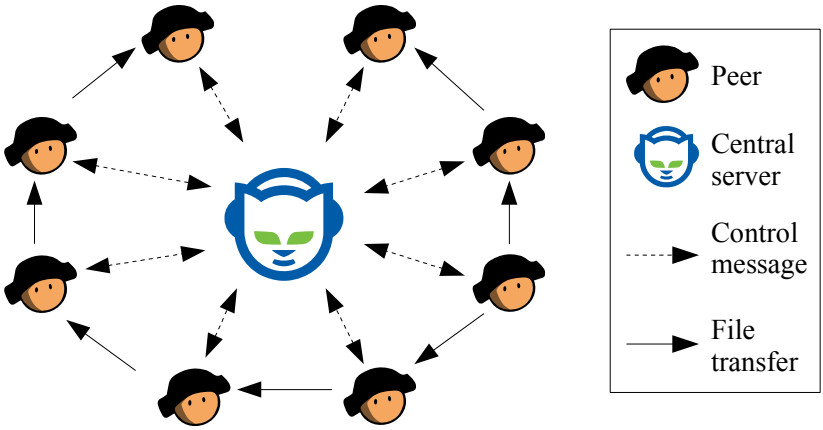

Figure 2.2: Napster P2P architecture

Napster was possible due to its main weakness – the presence of a central server.

Gnutella has been released in March 2000 [39], having been designed to overcome Napster's main weakness by adopting a fully-decentralized (i.e. pure-P2P) architecture, as pictured in Figure 2.3. Gnutella is classified as an unstructured P2P system, since peers are connected to a random set of other peers known as *neighbors*. A flooding technique is employed on the overlay for keyword search and obtention of a set of peers holding a particular file. Flooding consists on sending query messages to all neighbors, which in turn forward the query to their neighbors, until a maximum *hop count* is exhausted. Reply messages containing peers which are providers for the file sought after are returned to the original peer. This simple mechanism is effective for locating highly-replicated data and is resilient to churn [80], that is, peers joining and leaving the system with a high frequency, which is expected in a P2P system, since peers are in fact users that are not expected to always remain connected. The main drawbacks of Gnutella are (a) that flooding does not scale well, as the number of query messages grows exponentially to the number of connected peers, (b) that it may not find rare content, since flooding reaches a limited number of peers,

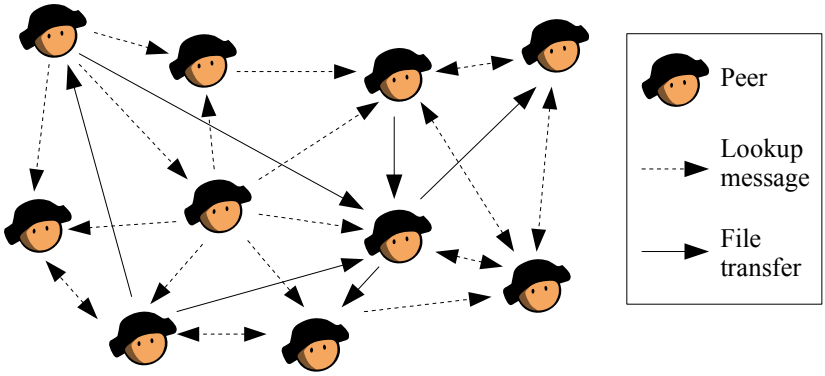

Figure 2.3: Gnutella P2P architecture

and (c) the increasing number of peers that refuse to contribute to other peers, namely *free riders* [4].

Released in July 2001, BitTorrent [9, 124] brought important innovations to the field. The system is designed to discourage free riding by implementing an incentive mechanism based on tit-for-tat (TFT) [6, 23], in which peers prioritize uploading data to peers that have provided most data in the recent past. Files are split into small (typically 256-kByte) chunks, to decrease overhead and improve distribution of large files, since chunks are downloaded in a rarest-first [69] order that increases chunk availability with high churn. BitTorrent's architecture follows a hybrid P2P architecture, as exemplified in Figure 2.4, employing servers – called *trackers* – to provide content to peer mapping, but unlike Napster, the system is not limited to a single server; multiple servers can be used simultaneously, which improves system stability and scalability.

BitTorrent trackers may also be designed with a pure P2P architecture by using Distributed Hash Tables (DHTs) as a substrate [80]. DHTs provide scalable, fully-distributed algorithms to locate, store, and retrieve key-values mappings, which are useful, for instance, to implement keyword search and provider look-up functions required by P2P file sharing systems. Such systems are considered structured P2P systems, since connections among peers are well-defined. Though there are several DHT implementa-

tions, such as Chord [110], Pastry [99], and Kademlia [84], they all share the important property of scalability, since all operations result in a number of messages only logarithmically proportional to the number of peers in the system.

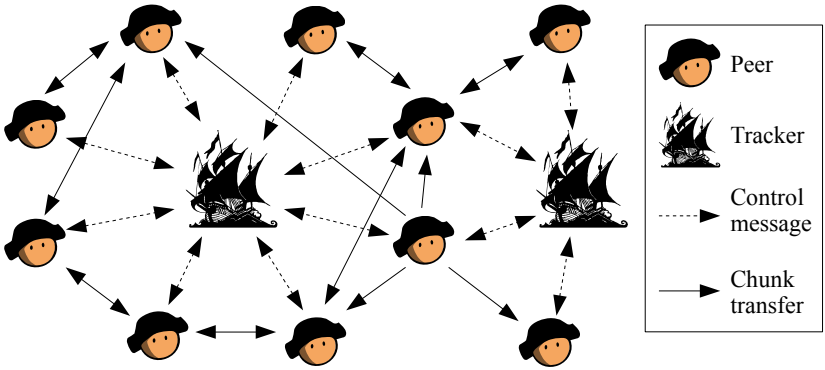

Figure 2.4: BitTorrent P2P architecture

Several other P2P systems that have emerged in the early 2000s have as well attracted numerous users, among which are FastTrack [73] and eDonkey [52, 116]. P2P traffic has been measured to account for the majority of all Internet traffic today, and the bulk of this traffic stems from BitTorrent [22, 47, 102]. Besides having attracted great attention from users and developers, P2P systems have received large interest from the research community as well, with several authors focusing, for instance, in overcoming malicious peer behavior and improving security [13, 35, 79, 117], exploring traffic localization [7, 58], increasing ability to cope with churn [98], and reducing the dependence on centralized trackers via DHTs [26] and the peer exchange (PEX) [92] protocol.

2.3 P2P Multimedia Streaming Systems and Protocols

While P2P file-sharing systems support the exchange of any file type, they are primarily used for exchange of complete multimedia files [124]. They, however, do not support multimedia *streaming*, that is, simultaneously

downloading and decoding (i.e. watching) content. A P2P system for multimedia streaming differs significantly to one designed for file sharing, due to its inherent asymmetrical nature – streams are bound to travel a single direction in the overlay – and its stringent time constraints – content needs to be located and downloaded [15, 49, 77, 87].

Employing a P2P architecture for multimedia streaming brings both advantages and challenges. A C/S architecture using unicast is technically simpler, but shows scalability problems and high cost, particularly with bandwidth. IP multicast, first standardized in 1985 [29] to solve those problems, has suffered a multitude of updates, e.g., for inter-domain routing [30], yet it has failed to achieve a global deployment due to both technical and commercial reasons [32]. Thus, several P2P architectures to multimedia streaming have been proposed; some systems already gather a significant number of users [53]. While P2P traffic has the largest share of Internet traffic today, Internet video streaming traffic has tripled in 2010, making Internet video become 40 percent of consumer Internet traffic [22].

Multimedia streaming is classified here into the three categories, according to how content is distributed and consumed by users: live streaming, Video-on-Demand (VoD), and time shifting (TS). Although differences might seem minor from a user's perspective, technological differences are fundamental [87]. The following subsections provide a technical overview on each category.

2.3.1 P2P Live Streaming

P2P live streaming systems offer the functionality of regular television and radio broadcasts over the Internet, thus supporting a much higher number of simultaneous transmissions (channels), since it uses the Internet as medium instead of radio waves, which are highly regulated. The set of available user operations in live streaming is small, since only the live stream is propagated on the overlay; video cassette recorder (VCR) operations, such as pausing, rewinding, and fast forwarding, are unavailable. In live streaming, all peers are interested in downloading and playing back the livest possible content within a channel, that is, approximately the same data [83];

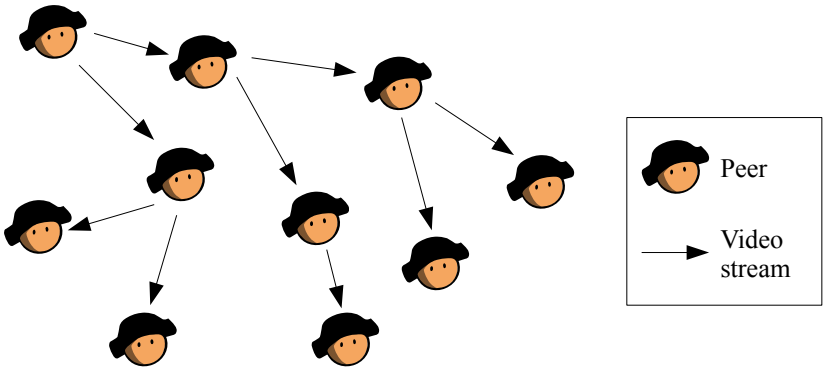

Figure 2.5: Single-tree P2P video streaming architecture

in addition, they cannot download at a rate faster than the stream is generated [87]. These are common assumptions present in systems designed for providing live multimedia streaming [103].

Early P2P live streaming systems have adopted a single-tree-based overlay topology, rooted at the stream source – the *peercaster*. Every peer receives the stream from a single parent and pushes it to a set of children, as depicted in Figure 2.5. The architecture is similar to IP multicast; examples are Overcast [62] and ESM [21]. Although it uses simple concepts, the single tree topology has intrinsic drawbacks that have prevented a large deployment from becoming reality. Upload capacity is heavily underutilized, since leaf nodes cannot provide bandwidth to other nodes and there is only a single stream to be distributed [17]. It also does not perform well under high levels of churn [77], as a whole sub-tree is negatively affected every time a peer joins or leaves the system.

In an attempt to overcome these problems, a multiple-tree overlay topology has been proposed; SplitStream [17] is a popular example. The original stream is divided into substreams using multiple description coding (MDC) [40], such that peers can continue playing the video stream even if some substreams are missing. As pictured in Figure 2.6, each substream is then pushed through a different tree, ameliorating the leaf-set problem. It also reduces the impact of a peer leaving the network, since peers can

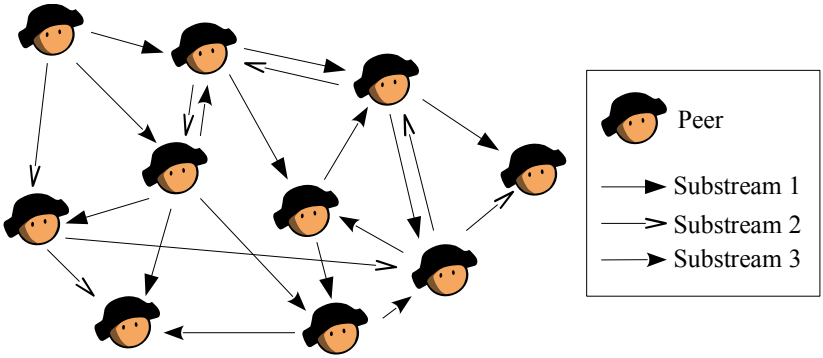

Figure 2.6: Multiple-tree P2P video streaming architecture

continue playing the stream without interruptions even with some missing descriptions, despite the degraded image quality. The multiple-tree topology, though, also has fundamental problems. The tree structure limits the formation of the overlay, resulting in sub-optimal resource allocation. Besides, its tree maintenance algorithm is expensive in terms of time and network overhead and does not perform well with variable network conditions, since background traffic and congestion still produce a cascading effect [82].

Most modern commercial P2P streaming applications, such as SopCast [107], PPLive [96], CoolStreaming [71], and PRIME [83] adopt the mesh overlay topology [81, 82]. The video stream is divided into chunks that are routed through the overlay with no fixed structure, and a gossip protocol is employed so peers communicate which chunks they hold. The topology, exemplified in Figure 2.7, shows excellent resilience to churn and good scalability, at the same time being very adaptable to varying traffic conditions, since a block that is not downloaded quickly enough may be promptly retried from another peer. However, these advantages come with a higher overhead, due to the gossip protocol and the fact that each block must be pulled (that is, requested) individually [82].

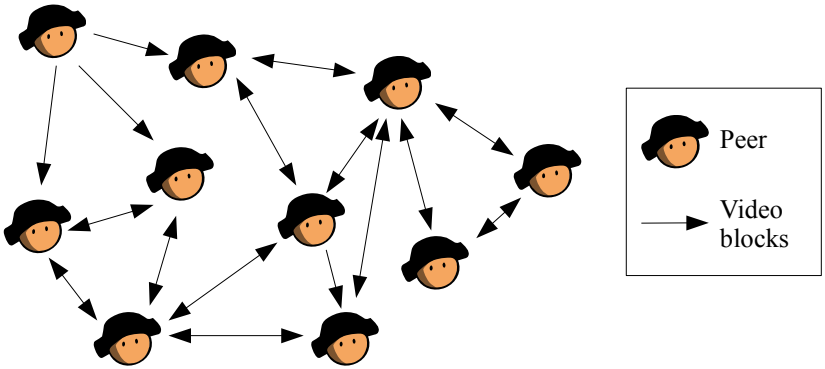

Figure 2.7: Mesh-pull P2P video streaming architecture

A hybrid push-pull topology has also been proposed [125], which attempts to combine the advantages of both tree and mesh. It consists on using a pull-based topology initially, then switching to push. Such systems are yet to show a large deployment on the Internet.

Independently from the type of high-level structure used to distribute the video stream, end-user QoE is still not satisfactory, with start-up delay ranging from several seconds to a few minutes for live channels [53, 77].

2.3.2 P2P VoD

P2P VoD systems are appropriate for distribution of pre-recorded content, such as movies and TV shows, because they allow users to select programs to watch them from the beginning at a convenient time. VCR operations are possible, depending on the specific implementation, allowing users to pause playback, as well as skip to different playback positions [19]. A common assumption is that content has been pre-encoded and is of known size.

From an architectural point of view, differences between live streaming and VoD systems are numerous [87]. In comparison with P2P live streaming, P2P VoD offers on one hand a lower opportunity for collaboration, since users playback positions are more diverse. Hence, it is much more difficult to alleviate the bottleneck at the peercaster's upstream link [60]. However, downloaded content might still be interesting to other users in

the future, so they can be stored, which increases their availability, while in live streaming, no peer is ever done downloading. A further important difference is that, on live streaming, the download rate is limited by the rate at which the stream is generated at the source, while VoD offers the ability of buffering more content ahead of playback. Overall, there exists currently far fewer widely-deployed P2P VoD than P2P live streaming systems [19]. One important reason is the support for VCR operations, which increases overlay dynamics.

P2Cast [42] an example of an early P2P VoD system that has adapted a tree structure to provide a VoD service. Peers are positioned in different trees, according to their arrival time. Video streams that are not available via any of the trees are downloaded as *patches* from any peer. Similarly to tree-based schemes, P2VoD [34] clusters peers according to their joining times. These systems, however, suffer from the same problems that hinder tree-based live streaming systems, namely waste of upload bandwidth and low resilience to churn.

The mesh-pull topology has as well been adopted by systems such as BiToS [118] and BASS [28], which aim on modifying the BitTorrent protocol to support VoD by, for example, modifying block and peer selection algorithms for in-order downloading and locality-awareness. RINDY [19] and VMesh [122] focus on allowing VCR operations on a P2P VoD system, the former by organizing peers in concentric rings according to the similarity of the content they hold, and the latter by considering a fully-distributed scenario. PPLive [96] is a proprietary, China-based video streaming and distribution system that has published [60] an insight into algorithms and usage of their widely-deployed system.

2.3.3 P2P Live Streaming with Time Shifting Support

The idea of a P2P live multimedia streaming system that supports time shifting is relatively new. Related work [31, 37, 43, 65, 68, 74, 75] has mostly relied, instead, on the separate and independent distribution of live and time-shifted streams. The seamless transition between live and time-shifted operation, as provided by LiveShift, has remained elusive. Further differ-

ences between these publications and the present one can be found in several aspects, including architecture and evaluation metrics. [31] is interested in evaluating the distribution of data availability and expected number of available peers, not the QoE obtained by users. [37] is very short and presents no insights or evaluation. In both [74] and [75], evaluation is very limited regarding QoE, since the impact of a higher number of failed requests is not clear – in addition, in this work peers need to store streams they never watched, only to be able to serve them to other peers. [43, 65, 68] are concerned with a centrally-controlled system for the sole distribution of time-shifted streams. [65] investigates scheduling schemes that supporting HD-quality streaming on small-bandwidth scenarios. [68] is concerned with evaluating replication schemes in a fully-cooperative environment. [43] describes its architecture and policies used for peer selection and in a closed, centralized, fully-cooperative environment. Though these research papers do not envision a fully-distributed, integrated P2P distribution of live and time-shifted multimedia streams, they do provide interesting initial insights in providing a P2P time-shifted multimedia streaming service.

Both SPPM [88] and PACUS [76] unify live and time-shifted streaming as envisioned in LiveShift, but suggest a hybrid P2P architecture, with a central entity holding a complete view of which stream segments every peer has, like Napster. Thus, this may become a single point of failure and result in scalability problems when the number of peers is very high, there are many streams, there is a flash crowd, or churn is high. In addition, it requires additional infrastructure (the central tracker) to be in place, which is costly to be maintained, thus particularly undesirable in a scenario with small peercasters, e.g., users broadcasting from their homes. The first of these, SPPM, adopts a multiple-tree overlay topology, which has been shown to perform badly with dynamic network conditions and churn [82], when compared with a mesh topology. Such conditions, though, are exactly what is expected when users are allowed to both switch channels and rapidly change their media playback position. Further, the evaluation of SPPM relies on simulations in which every peer is always able to upload at a rate twice the bit rate of the stream. This kind of reliable overprovisioning, while simplifying many problems, is unrealistic in a P2P envi-

ronment where bandwidth availability is expected to fluctuate. The second of these works, PACUS, presents a model based on multiple-interval graphs and several optimization strategies that can be used by a central tracker with a global view. Peers are assumed to always transfer complete 1-minute-long chunks to each other, which is problematic when peers need to download different chunks from different peers, each at speed lower than the bit rate of the stream, and combine them on time for playback. It is probably due to this fact that their evaluation only considers peer capacities which are integer multiples of the stream bitrate. Finally, the results presented concern only the reduction of bandwidth at the provider side, which are a natural consequence of using a P2P architecture, but overlook the issue of user experience.

NPR [89] focuses on pre-fetching strategies for a fully-cooperative mesh-based P2P live video streaming with support for time-shifting. The work is complementary to the proposed in this thesis, as it is concerned with some of the policies identified – block selection, upload, and download peer selection policies. However, some simplifying unrealistic assumptions are made by the authors, namely (a) that every peer can always upload at least at the bitrate of the live stream, (b) full chunks are always successfully transferred, (c) the server has unlimited upload capacity, thus user experience is always perfect, and (d) peers always send truthful reports including which chunks are urgent. The paper is not interested in user experience, but solely in reducing server bandwidth consumption. In this sense, the reference implementation of LiveShift may be useful.

A classification of different multimedia streaming systems according to category and architecture type is given in Table 2.1. The LiveShift system primarily fills the gap of offering a fully-distributed service of live and TS streaming.

Furthermore, Table 2.2 summarizes related work focusing on an integrated approach of live and TS multimedia streaming. LiveShift is distinguished, not only by adopting a fully-distributed architecture, but as well by its usage and evaluation assumptions. Peer upload capacities are considered the most scarce resource in such systems, due to asymmetric band-

Table 2.1: Selection of major multimedia streaming systems according to category and architecture

Category	*C/S*	*Hybrid P2P*	*Fully Distributed*
Live	Zattoo	SopCast, PPLive	PRIME
VoD	NetFlix, Joost, YouTube	RINDY, BiTOS, PPLive	VMesh
Live + TS/VoD	N/A	SPPM, PACUS, NPR	**LiveShift**

width provisioning by operators, which in fact drives development of P2P systems. The assumption on those capacities, thus, dramatically influence results obtained by measurements. As seen on the table, related work has, until now, assumed a rather optimistic distribution of upload capacities, not only assuming that every peer is always able to upload the stream to at least one other peer, but also assuming a constant rate. In reality, some peers might be able to upload only a fraction of the stream, and the rate may vary due to congestion and competition from other traffic flows. Further, SPPM uses peak signal-to-noise ratio (PSNR) [61] as a QoE metric, which is an improvement over other works that do not consider user metrics; however, while PSNR is an appropriate metric to compare different video encoding and decoding algorithms, it is not applicable in scenarios where stalling is possible, which is the case with most P2P systems. In contrast, LiveShift uses a complete software implementation to evaluate user experience in realistic scenarios, including channel switching at a high frequency and a majority of peers with low upload capacity, starting from a fraction of the stream bitrate. Further, performance evaluations include metrics that relate directly to user experience. Finally, LiveShift's evaluations are performed with full implementations, allowing a more realistic insight into inter-dependencies among different factors that influence its performance.

Table 2.2: Comparison of P2P systems that support integrated live and time shifting streaming. x stands for the bit rate (video + audio) of the stream.

Work	*Evaluation Method*	*Peer Upload Capacities*	*QoE Evaluation*
SPPM	simulations	always $2x$	yes
PACUS	simulations	only integer multiples of x	no
NPR	simulations	at least $1x$	no
LiveShift	full implementation and emulations	at least $.5x$	yes

2.4 Quality-of-Service and Quality-of-Experience

Quality-of-Service (QoS) and Quality-of-Experience (QoE) are two related terms commonly used for defining metrics for evaluation and comparison of multimedia streaming systems. This section provides an overview on both and argues for the QoE metric used in the evaluation of LiveShift contained in this thesis.

A precise definition of QoS depends on context [36]. In literature, it may refer to user perception of the service, but also to the set of connection parameters required for a particular service quality to be achieved [108]. According to [3], QoS is divided in three categories – intrinsic, perceived, and assessed. Intrinsic QoS stems purely from technical aspects, including transport protocols and provisioning of network access, terminations, and connections, mostly made of packet-level metrics such as bit rate, packet delay, jitter, and packet loss rate. Perceived QoS manifests user experience, being characterized by the user expectations and observed performance. The assessed QoS measures the decision of the user to remain with a service provider or not, encompassing quality but also other aspects, including service price and other similar offers [108]. Still, the term QoS is often used in literature referring exclusively to intrinsic QoS.

QoE has very recently attracted increased attention from academia [57], and encompasses mostly perceived quality metrics. Two basic measurement types are present – subjective and objective. A typical subjective QoE metric is the mean opinion score (MOS), which is directly determined by humans rating a media transmission. Although it is convenient to have a single number summarize user experience, a multitude of both internal and external factors can influence this metric – quality of the transmission, interest and expectations of users, type of content, properties of display and light conditions, and cultural and psychological background of users are only a few important factors. Besides, experiments are very time-intensive and require hundreds of probands, thus being very costly. Because of these drawbacks, objective QoE has been defined as automated procedures involving algorithms that approximate subjective QoE without requiring active ratings by users [36, 56]. These works, though very interesting, consider only short movie streams. The full ramifications of playback in LiveShift, which encompass factors illustrated in detail in Section 4.1 – particularly playback lag that increases with larger start-up delays and interruptions during the streaming session (stalling) and possibility to skip content to decrease playback lag avoiding interruptions, combined with playback that may be live (or close to live in case of TS) – are yet to be addressed by an objective QoE model.

Hence, in this thesis, a reasonable assumption that the used application-level QoS metrics – mainly playback lag and number of skipped blocks – are tangible, measurable, comparable, and relate directly to QoE. Further, these metrics are independent of video encoding and decoding algorithms, user expectation, and environmental, sociological, and cultural factors. Based on these facts, application-layer QoS metric, namely playback lag and share of skipped blocks, are used in this thesis as QoE indicator metric (*QoE metric*). These terms are thoroughly defined in Section 4.1.

2.5 Playback Policies

Although numerous research target P2P video streaming, most are concerned with different types of overlays (e.g., tree, mesh) [81], peer selec-

tion strategies [5], or incentive mechanisms for peer contribution [94]. Though the implementation of a playback policy is required in any video streaming system, it is often omitted. Those systems that do describe the adopted playback policy are introduced in this section.

The most popular P2P live video streaming applications, such as SopCast [107], are proprietary and do not disclose in detail their policies. Measurements [103] suggest that these systems, after performing initial buffering, employ a window that moves at constant speed, skipping all blocks that are not downloaded on time. The assumption is that, since streaming is live, maintaining liveness is more important than attempting to play every single block. It also helps keep peers mutually interested in a narrow content window. Such policy is also used on research papers that model live P2P systems [66].

For VoD, the assumption is frequently the opposite. Since liveness is not important, an intuitive policy would be, after performing initial buffering, stall every time a block to be played is missing. In a P2P system, though, such policy could cause playback to stall for a very long time in case there are a few very rare blocks. The VoD P2P client Tribler [86] addresses this issue by stalling playback only if less than 50 percent of a 10-second playback buffer is filled; otherwise, it skips to the next available piece.

The work presented in [87] also uses Tribler for VoD, but adopts a different policy. Playback is stalled until a 10-second-long buffer is filled and the remaining download time is less than the duration of the video plus 20 percent. The policy does not allow skipping.

Gridcast [20] is a P2P VoD system which playback policy consists on stalling if a block is not available at its playback deadline, and attempting to play it up to 10 times. If the block still has not been downloaded, playback skips to the next block. Initial buffering is 10 seconds, and each block is 1 second long.

LiveShift [46] adopts a unified policy for live and on-demand streaming which consists on skipping n contiguous missing blocks if and only if the peer holds at least $2n$ contiguous blocks immediately after those, otherwise stalling. It aims not to let peers stall for long in case only a few blocks are not

available from current neighbors, while skipping if there is a good chance of continuing playback without interruptions.

While all those publications discuss briefly the playback policy adopted, they do not offer a plausible proof or justification to why such algorithms and parameters were chosen in the foreseen scenarios, and, as described, are not comparable under a consistent set of parameters. Thus, the formal definition leading to a comparison of QoE metrics resulting from the adoption of each playback policy is highly needed, in particular in the environment proposed in this thesis, since it includes both live and TS streaming abilities.

2.6 Distributed Tracker Approaches

Different types of distributed trackers have been proposed and deployed so the tracker can benefit from P2P properties. They are divided into DHT approaches and gossiping. Both approaches have drawbacks that promote the development of a novel distributed tracker, as described in the next subsections.

Distributed hash tables (DHTs), such as KAD [26, 109], are able to map *keys* (e.g., content) into *values* (e.g., providers). DHT functionality needs to be modified to allow several values to be added to a single key and to return a random subset of those values when queried. DHT-based trackers are pull-based, which allows a peer to retrieve a new set of providers as soon as and only as long as it is needed. The fact that a random subset is returned, regardless of which providers the requester already has already obtained, reduces its efficiency, since a large amount of traffic may be used to transfer useless providers.

Another important problem is load balancing. Content popularity resembles a power-law distribution, as seen in Figure 2.8, which displays how many peers are interested in obtaining or providing torrent files (a collection of files) at The Pirate Bay [47]. Since DHTs keep a constant number of replicas per key-value pair, peers responsible for tracking popular content have a much higher load than others.

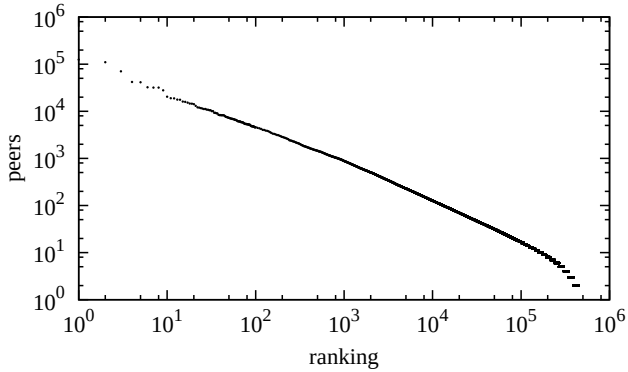

Figure 2.8: Popularity of torrents at The Pirate Bay

There is extensive academic work in improving load balancing in DHTs, dating back to more than 10 years [27, 97]. However, as of today, it is still an unsolved problem [59]. The main drawback of several approaches [27, 97, 111] is that they consider load in DHTs merely the number of key-value pairs stored by each peer, disregarding access patterns. This is a problem for P2P trackers due to the expected power-law distribution of content popularity. While key-value pairs in P2P trackers occupy little disk space, upload bandwidth is limited.

A common approach is to use the concept of virtual servers [59, 97, 105, 121], which allows storage responsibility of key-value pairs to be exchanged among peers. Peers are classified into heavy peers (ones that are overloaded) and light peers (underloaded). These works differ by the way in which virtual servers are transferred from heavy to light peers, and how each category is identified. Using a virtual server approach would certainly improve load distribution in P2P trackers, but it also bring undesirable side effects. As each virtual server contains several key-value pairs, as soon as a single pair becomes popular, all other ones are as well replicated, even if they are not popular, which is inefficient. Besides, a movement cost [59] is always associated with each virtual server replication, even if those are minimized [105]. Hence, implementations of such techniques are not widely used in P2P trackers.

Finally, since those DHT load balancing schemes are generic in nature, they do not explore properties that are present when using DHTs specifically as P2P trackers. The contribution opportunity lies in the fact that movement cost in tracker approaches can be eliminated by allowing peers to cache information they would nonetheless retrieve.

Another way of designing a distributed tracker is using a gossip protocol, such as Peer Exchange (PEX) [33, 44, 92], which is implemented by several BitTorrent clients as an extension of the original protocol. Using PEX does not eliminate the need for a tracker, since every peer must still contact the tracker (C/S or DHT) in order to receive its first provider list.

Though different implementations of PEX exist, their main idea is that peers keep their neighbors informed about their current neighbor set. This is generally done by periodically (e.g., once a minute) sending messages containing sets of added and removed neighbors [10] to every neighbor. When new providers are needed, peers select those peers that appear least frequently as their neighbors' neighbors. The reason why less popular ones are chosen is that those are probably newly arrived and need new neighbors. Load balancing is expected to be improved in PEX, since every peer is responsible for sending regular update messages. But, since gossip protocols are push-based, they need to consider a trade-off on the frequency of gossip messages sent. If sent less frequently, the information is spread more slowly, which may be troublesome, especially for delay-sensitive applications, such as video streaming. If sent more frequently, efficiency decreases, as information will be more redundant.

Table 2.3 shows a comparison between the expected efficiency and load balancing properties of DHT, PEX and B-Tracker, summarizing properties described. Efficiency refers to the traffic generated per peer to spread the knowledge of providers, while load balancing refers to how well the traffic is distributed among the peers.

Table 2.3: Distributed tracker related work comparison

Approach	*Efficiency*	*Load Balancing*	*Push/pull*
DHT	–	–	Pull
PEX	– –	+	Push
B-Tracker	+	++	Pull

2.7 Contribution Opportunities

While the field of P2P multimedia streaming has received significant attention from the research community, the related work research present in this chapter has revealed the following:

1. current approaches that support the seamless integration of live and time-shifted video streaming under a single P2P protocol do not employ a fully-distributed architecture, and use rather unrealistic evaluation methods to measure their performance, i.e., not including user experience metrics and relying on simulations with unrealistic peer upload capacities;

2. though a playback policy is a mandatory part of any P2P streaming system, several publications do not disclose enough information about the one used, and the few publications that do define a playback policy, do so arbitrarily, without scientific support; and

3. current distributed P2P tracker technologies lack a combination of fair load balancing and high efficiency properties as content popularity is highly unequal.

Having identified the above shortcomings, and in direct relationship with the four observations made in Section 1.6, the following opportunities for scientific contribution in the area of P2P multimedia streaming have been revealed:

1. unification of P2P live and time-shifted video streaming under a novel, single protocol, using a fully-distributed architecture, which

employs a mesh overlay topology to achieve better resilience to churn and users changing channels and position in the time scale, allowing the evaluation of user experience under realistic assumptions, including peers with upload capacities lower than the video stream rate;

2. the unification of live and time shifting functionality calls in particular for the analysis and comparison of QoE metrics of a large set of playback policies under comparable conditions, to derive which are most suitable for live and on-demand scenarios;

3. development of a novel distributed tracker technology that improves both load balancing and efficiency using a fully-distributed architecture; and

4. a reference client implementation that allows obtaining the above-mentioned realistic user experience metrics, besides allowing public demonstrations of the application, facilitating comparative testing of different policies by the research community, and enabling the release of the application as open source software.

Hence, chapters 3 to 6 present a detailed description of those contributions indicated above.

3

LiveShift Architecture, Protocol, and Policies

Allowing the seamless integration between P2P live multimedia streaming and time shifting as envisioned in this thesis was a relatively unexplored research field before this thesis work. Section 2.3.3 has identified that current approaches suffer from major drawbacks that prevent them from becoming reality. Thus, this chapter describes the LiveShift architecture, a flexible mesh-pull protocol that supports the envisioned use case, a set of preliminary policies to be used with the protocol, and presents evaluation results. It is important to notice that the LiveShift system has been fully implemented, allowing the evaluation of the protocol and involved policies under highly realistic conditions, including a high ratio of peers with lower upload capacity than the stream rate, channel switching at a high frequency, and churn, that is, peers joining and leaving the system.

An initial version of LiveShift, presenting the envisioned use case and including basic architecture, has been published as a demonstration in [48]. A complete specification of architecture, protocol, policies, and evaluation results have been published as both a research paper [46] and a technical report [45], the latter containing additional details and evaluation results.

The two main contributions of this chapter are (a) to propose a new, fully-distributed, P2P streaming, mesh-pull protocol, which is suitable for live streaming, time shifting, and VoD, and (b) to propose, discuss, and analyze initial policies that influence the protocol usage. While this chapter does not claim that the presented policies are optimal, it introduces the main challenges and trade-offs involved when designing those in a system in which live and on-demand streaming are indistinguishable. Protocol and policies are evaluated using traces from a real IPTV system to model peer behavior, namely channel switching and holding times. Results are measured in terms of QoE metrics, such as playback lag, that are important for the end user, as discussed in Section 2.4. This is the first fully-decentralized, mesh-pull protocol designed for both live streaming and VoD, and which has been tested to include the effects of peer channel browsing behavior.

The rest of this chapter is organized as follows. Section 3.1 presents fundamental design objectives of the LiveShift system. Section 3.2 introduces layers and components which are part of LiveShift's architecture. The protocol is described in Section 3.3, and policies in Section 3.4. Section 3.5 presents evaluation scenarios and results, and Section 3.6 summarizes the main contributions of this chapter.

3.1 Design Objectives

The main design objectives of the LiveShift protocol and system are the following. These have been chosen based on typical P2P characteristics [85].

1. **Free Peercasting:** Any peer is able to publish a channel, therefore becoming a *peercaster*;

2. **Scalability:** The approach shall scale to a high number of peers, even when several of them are only able to upload at a fraction of the bit rate of the video stream;

3. **Robustness:** The system must tolerate churn in form of peers joining and leaving the system and switching both channels and positions in the time scale;

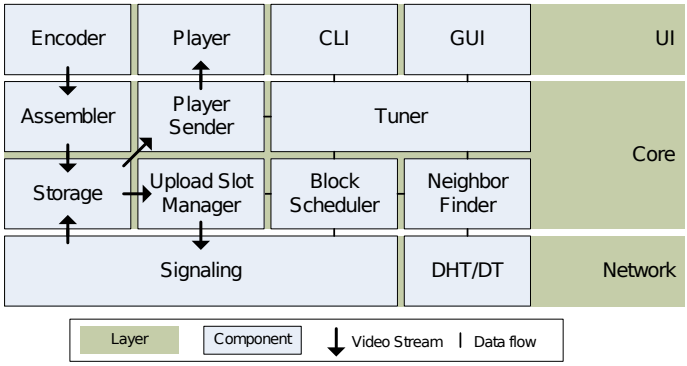

Figure 3.1: LiveShift top-level architecture

4. **Full decentralization:** In order to allow any peer to publish a channel without deploying a large infrastructure and to fully benefit from P2P properties, no central entities shall be present – except peercasters, a single point of failure for the live stream of the channel they originate; and

5. **Low overhead:** Video streaming is very bandwidth-consuming and sensitive to delay, therefore network overhead introduced must be low.

3.2 Main Components

Figure 3.1 shows, in a simplified sequence diagram, the components of the top-level LiveShift architecture, which is split into three layers. The Network layer contains the Distributed Hash Table/Distributed Tracker (DHT/DT), as well as the Signaling/Video component. The DHT/DT component stores network-wide persistent information in a fully-decentralized manner, and its functionality are provided by the TomP2P [115] library. The Signaling/Video component handles direct message exchange between peers, including video block transfers.

The Core layer contains all main software functionality. The Storage component stores video blocks, which contain parts of the video stream.

Blocks are stored into the Storage component from the Assembler if the peer is a peercaster, otherwise, they are input from other peers via the Signaling/Video component. The Upload Slot Manager component manages creation, granting, and verification of upload slots. Its policies are of fundamental importance to the distribution of the video stream, since it selects how many and which peers will be granted upload slots, thus receive video blocks, and distributes the available upload bandwidth among upload slots. The Block Scheduler selects next blocks to be played and sends messages requesting blocks to appropriate peers. The Neighbor Finder maintains and manages sets of candidates C and neighbors N, querying the distributed tracker when necessary. The Assembler receives the encoded video stream from the Encoder and partitions it into blocks and segments. The Player Sender obtains blocks from the Storage, joins the video data contained in them, and sends it to the Player at the right rate. The Tuner receives commands from the UI about which channel and starting time to tune into. It, then, initializes and coordinates the Neighbor Finder, the Block Scheduler, and the Player Sender.

The User Interface (UI) layer manages machine-human interfaces, encodes and decodes video. The Encoder is responsible for receiving signal from a video capture device or file, encode them properly, and output them to Segment Assembler. LiveShift uses the VLCJ library [119] as encoder. The player is responsible for decoding the video stream and playing it back to the user. VLCJ is also used for this purpose. The CLI (Command-Line Interface) is used to control automated experiments from the command line. The Graphical User Interface (GUI) is operated by users in demonstrations and trials.

3.2.1 Segments and Blocks

The proposed use case gives users the possibility of switching channels and time shifting. LiveShift adopts the mesh-pull approach [53], which adapts better to dynamic network conditions and churn when compared to tree protocols [82]. Mesh-pull divides the stream into chunks that are exchanged between peers with no fixed structure. Two levels of chunking

are used – a *segment* is an addressing entity, which is made up of several smaller *blocks*.

LiveShift addresses and identifies segments and blocks based solely on time – they are both of well-known fixed time-based length. This makes it trivial to discover which block and segment contain the part of the video stream to be played at a given time. This is especially useful for live streaming, since it may be difficult to predict at which bit rate the video will be generated in the future. It also makes it simple for peercasters to change the bitrate of the offered channel, or even offer variable bit rate (VBR) streams to save bandwidth in sequences that can be better compressed. Each segment is uniquely identified by a *SegmentIdentifier*, which is a pair (*channelId*, *startTime*) announced on the tracker by peers which offer video blocks within a segment. Blocks are small-sized, fixed-time video chunks, and are the video unit exchanged by peers. Discussion and definition of segment and block lengths are performed in Section 3.4.1.

Differentiating segment and blocks is important to reduce the number of tracker operations (due to larger segments), while allowing peers to download blocks from several other peers, recovering quickly if any fails (due to smaller blocks). Thus, only segment availability is announced on trackers; block availability within a segment is queried directly between peers, considering that it is more sensitive to timing and synchronization issues.

3.2.2 Distributed Hash Table and Distributed Tracker

LiveShift uses a distributed hash table (DHT) to store the channel list and individual channel information. There are three DHT operations available: `GetChannelList` retrieves a list with all available channels and *channelIds*, `PublishChannel` and `UnpublishChannel` creates and removes a channel, respectively. A possible enhancement would be adding an electronic program guide (EPG) to map programs to (*channelId*, *startTime*), achieving VoD-style program browsing.

The tracker is responsible for mapping segments to a set of providers – peers that hold at least one block in the segment. LiveShift uses B-Tracker, a fully-distributed tracker (DT), maintained by all peers in

the system, and described in detail in Chapter 5. B-Tracker improves scalability and load balancing, while avoiding a single point of failure and reducing the infrastructure deployed by peers. There are three DT operations: `PublishSegment` is invoked by providers of a segment, `UnpublishSegment` is called by peers that have removed all blocks in a segment from their local storage, and `GetCandidates` retrieves a set with peers that provide a particular segment. The DT differs from the DHT, since its operations are invoked much more frequently. Therefore, every peer that is a provider for a certain segment caches the provider set obtained, which can then be supplied to other peers on request. This improves load balancing, which is of especial importance on popular segments. More information on the DT can be found in Chapter 5.

Since peers may leave the system unexpectedly, each tracker entry has a timeout value; thus, peers need to periodically refresh their content availability. A timeout value of 30 minutes is currently used. The timeout value is a compromise between overhead and chance of holding outdated information in the tracker, especially in the presence of churn. Even if a central tracker were in place, such a timeout value would still be important to remove outdated information.

3.3 Protocol Overview

The protocol is designed to be flexible by allowing the implementation of different policies for specific functions; a set of basic policies is proposed in Section 3.4. A key difference between LiveShift and other existing P2P systems and protocols regards symmetry of interest. Supporting VoD as envisioned in LiveShift requires that peers serve stored video blocks without necessarily being simultaneously interested in what the peer receiving the download has to offer, which brings the need for the separation between neighbors and subscribers as described below. This has implications with respect to incentive mechanisms which, although out of scope of this thesis, have been successfully approached in the literature [13, 67].

Figure 3.2 shows, in a simplified message sequence diagram, the steps taken by a peer to locate and download content; not every possible message

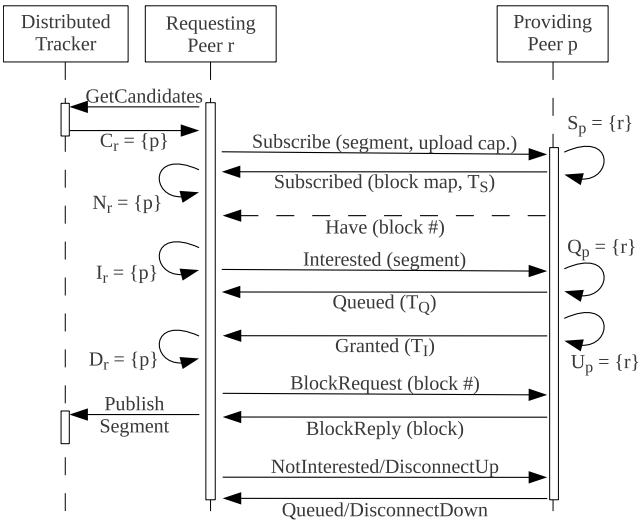

Figure 3.2: LiveShift protocol example message sequence diagram

is displayed for simplicity of representation. For nomenclature, please refer to Table 3.1. The complete message specification is found in Appendix A.

Figure 3.3 displays states and respective transitions that providing peers undergo at requesting peers. Conversely, Figure 3.4 displays those undergone by requesting peers at providing peers. Note that, on both state machines, the "Any State" state represents any other state in the diagram, and is used for clarity of representation. All these figures illustrate the protocol concepts introduced in the rest of this section.

A peer *r*, when entering the system, retrieves the channel list from the DHT. After having chosen a *channelId* and a *startTime* to tune into, *r* consults the DT to retrieve a set C_r of candidate peers (providers) that have advertised blocks in the corresponding segment. Another way of obtaining candidates is receiving `PeerSuggestion` messages, which contain suggestions for new candidates, directly from any peer. Peer *r* then contacts a number of candidates $p \in C_r$ by sending each a `Subscribe` message,

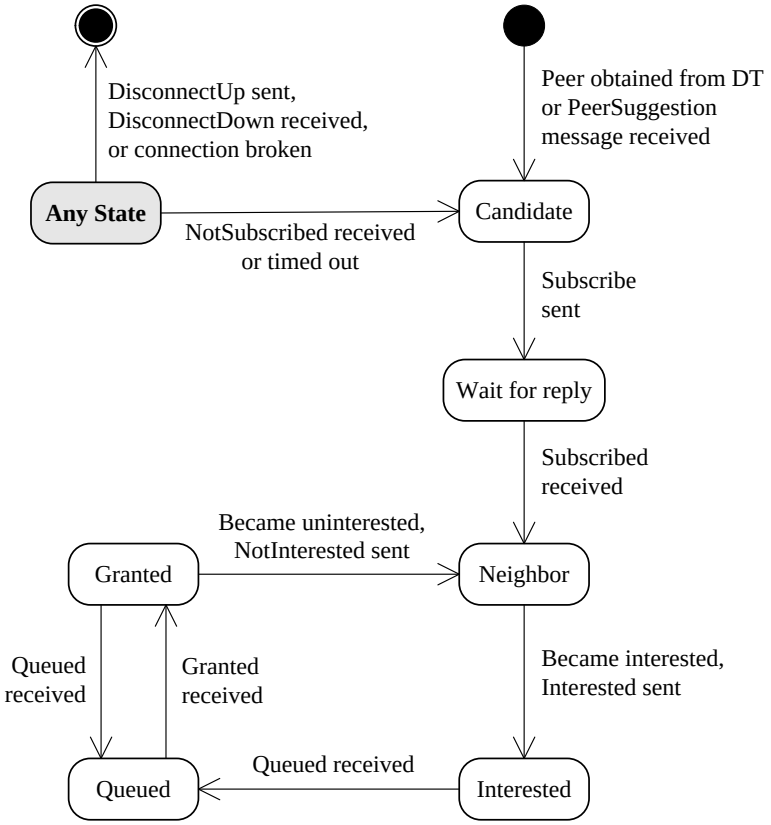

Figure 3.3: LiveShift protocol state machine for each provider, requesting peer view

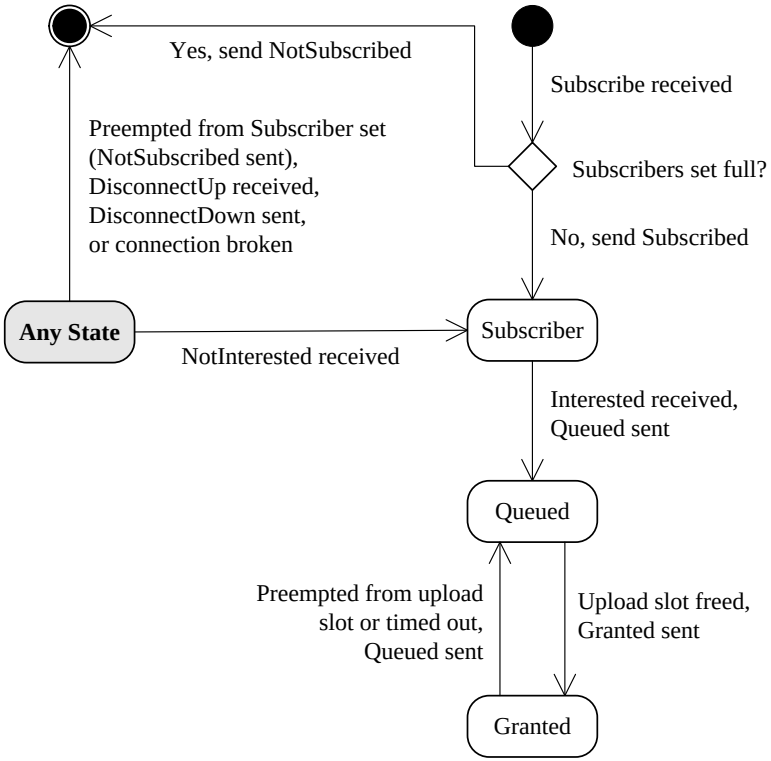

Figure 3.4: LiveShift protocol state machine for each requester, providing peer view

Table 3.1: LiveShift basic nomenclature

C_r	set of candidate peers that announce at least one block in a segment at which r seeks for blocks
N_r	set of peers in which r is subscribed to block map updates
I_r	set of peers in which r is interested, waiting for a slot
D_r	set of peers which are granting r an upload slot
S_p	set of peers that subscribed to block map updates from p
T_S	time limit a peer is allowed to stay in S
Q_p	set of peers that have manifested interest and are in the upload queue waiting to get an upload slot from peer p
T_Q	time limit a peer is allowed to stay in Q
U_p	set of peers a peer p is granting an upload slot to
T_I	time limit a peer is allowed to stay in U inactive (i.e. maximum time between two `BlockRequests`)
$\overline{X}$	maximum allowed size of set X
$\|X\|$	current size of set X

containing the *SegmentIdentifier* and a declared upload capacity. Verifying the correctness of the upload capacity is out of the scope of this thesis.

When a peer $p \in C$ receives a `Subscribe` message from a peer r, it attempts to place r in its subscribers set S_p. If $|S_p| < \overline{S}_p$, the subscribers set is not full yet, and peer r is sent a `Subscribed` message, with a block map indicating which blocks in the requested segment p holds and a timeout value T_S. Peer r will then be subscribed to receive updates to the corresponding block map via `Have` messages. If $|S_p| = \overline{S}_p$, p checks if there is another peer $q \in S_p$ that has lower priority than r (according to the policy used, *c.f.* Section 3.4.4). If so, it will be preempted immediately and removed from the set. Thus, either q or r will receive a `NotSubscribed` message. Limiting $|S_p|$ is important because if each peer has $|S|$ subscribers for a particular segment, the total number of `Have` messages sent for each new block in

the whole P2P network is $|S|^2 - |S|$. The constraint that a peer will not send `Have` to a peer which has reported to hold the block only reduces it to $(|S|^2 - |S|)/2$. Note that the proposed algorithm contrasts with existing mesh-pull P2P streaming ones in two important aspects; first, that the relationship between peers is asymmetric, and second, that immediate preemption causes a quicker reaction compared to re-evaluating peer selection at fixed time slots, reducing overall playback lag.

When peer r receives `Subscribed`, it adds p to the neighbor set N_r and must verify interest periodically by computing the intersection between blocks scheduled for downloading and blocks announced by p via its initial block map and following `Have` messages. If the intersection is not empty, r adds p to I_r and sends it an `Interested` message, which makes p add r in $Q_p \subset S_p$, the queue for peers waiting for an upload slot, and reply a `Queued` message, with a timeout value T_Q. On the contrary, when p has no more interesting blocks, r sends it `NotInterested` to be removed from Q_p.

Peer p has a number of upload slots $\overline{U}_p$, each of which is granted to an interested peer $r \in Q_p$. When peer r is granted an upload slot, it receives a `Granted` message, with an inactivity timeout value T_I, such that an upload slot that has not been used for T_I seconds is granted to another peer. Similarly to what happens in S_p, peers with higher priority may immediately preempt other peers from upload slots.

When r is granted an upload slot from p, it is allowed to send `BlockRequest` messages to p and receive video blocks in `BlockReply` messages. Each upload slot queues up to two `BlockRequests` at a time, to fully utilize its upload capacity, with no delays between sending a `BlockReply` and receiving the next `BlockRequest` message. This happens until (a) r sends either a `NotInterested` or `DisconnectUp` message, (b) p sends either a `Queued` message (if r is preempted) or a `DisconnectDown` message, or (c) r times out.

The two different types of disconnect messages reflect the asymmetry of interest present in the system. `DisconnectUp` is issued by a requesting peer r which is not anymore interested in downloading a particular segment, i.e. switching channels. A peer p that receives `DisconnectUp` stops uploading to r, removing it from S_p, Q_p, and U_p. `DisconnectDown`, in

contrast, is issued by a providing peer p that is leaving the system. It is similar to the `NotSubscribed` message, in the sense that it communicates that r has been removed from S_p, Q_p, and U_p, so it must remove p from C_r, N_r, I_r, and D_r. The difference is that the retry behavior of r towards p should take into account that it has actually left the system. Differentiating disconnect messages allows a peer that is switching channels or time shifting to change peers it is downloading from without breaking its uploads.

3.3.1 Peer Departure and Failure

Three mechanisms are present so the system reacts quickly to peers leaving unexpectedly or failing, namely timeout values, reporting of DHT routing errors, and ping messages. The timeout values T_S, T_Q, and T_I are defined to impose a limit on the resources taken by peers that leave the system unexpectedly. The latest received timeout value always overwrites all previously received ones.

Additionally, the DHT is able to inform LiveShift about routing errors. When a routing error occurs, a moving average for the failing peer is incremented. When the moving average exceeds a threshold (currently 4 over the last 5 seconds), the peer is removed from all sets, leaving space for other peers. This allows the system to react quickly to peers malfunctioning or leaving the network, since routing errors may occur before any message sent to the failing peer times out. The use of a moving average absorbs temporary or intermittent peer or network failures.

Finally, `PingRequest` messages may be used to test if peers are online. Peers must reply with a `PingReply` whenever they receive a `PingRequest`, otherwise they are considered as having failed.

3.4 Current Policies

As stated in Section 3.3, LiveShift protocol is flexible in the sense that it may be used with different sets of policies. This section focuses on the discussion of the engineering trade-offs embodied by the identified policies. These policies are simple enough to produce reliable results, yet complete

enough to give an accurate representation of the design space for streaming protocols capable of both VoD and real time operation. Finding optimal values, performing a parameter sensitivity study, and analyzing the main effects and interactions between parameters, are, though, out of scope of this thesis and left for future investigation. Table 3.2 compiles LiveShift's current policies, which are clarified in detail in the next subsections.

3.4.1 Length of Segments and Blocks

Larger segments mean less entries in the DT and less `Subscribe` and `(Not)Subscribed` messages, but reduce the chance of locating interesting blocks in peers, since in LiveShift peers announce a segment on the DHT when holding at least a single block in it. LiveShift currently uses 10-minute-long segments, which have shown to produce good results. This way, for instance, when watching a two-hour-long movie, a peer will announce 12 segments.

To minimize delay, blocks must have a small size, since they can only be uploaded after they are completely downloaded. Yet, the smaller they are, the larger is the overhead with headers, block maps, and `Have`, `BlockRequest`, and `BlockReply` messages. A block length of one second has shown to provide a good compromise, since a peer is still able to download each one quickly from different peers and combine them in a short time period. Hence, a segment contains 600 blocks.

3.4.2 Block Selection and Rescheduling

Another important decision is how many video blocks peers try to download ahead of the playing time. Downloading many blocks ahead may decrease the probability of a block not being present at playback time. It may not, though, be always desirable to read ahead as much as possible, since doing so may consume resources from other peers that could be used to send blocks to the community. LiveShift selects the next 15 missing blocks for downloading, counting from the current playback position, with two limitations: (a) at most 30 blocks ahead of the playback position are selected,

Table 3.2: LiveShift policy parameter overview

Policy Parameter	*Value*
Length of Segments	10 min
Length of Blocks	1 s
Block Selection	next 15 missing blocks, at most 30 ahead of playback position
Block Rescheduling	4 s for first block, then twice moving average
Candidate and Neighbor Selection	40 random peers from DT as candidates, up to 15 peers as neighbors, preference: (1) least amount of `Subscribe` sent, (2) highest amount of blocks provided, (3) random
Candidate and Neighbor Removal	5 Subscribe messages sent without having provided any blocks or 8s behind playback position
Subscribers and Upload Slot Selection	$\overline{S}_p = 5 \cdot \lvert U_p \rvert$, $\lvert U_p \rvert$ = at least full rate to each slot, preference: (1) higher upload capacity, (2) highest amount of blocks received, (3) random
Timeout Values	$T_S = 5s$ $T_Q = 10s$ $T_I = 4s$
Playback	$n = 2$
Storage	LRU, up to 2 h

and (b) blocks that were not yet produced because their timestamp is in the future are not scheduled for download.

Peer r downloads each selected block in ascending chronological order from each peer $p \in D_p$ if p announced to hold the missing block. Another option would be downloading rarest blocks first, but since a short time period between playing time and current time is a possibility, the policy is not used at the moment.

Since peers may fail unpredictably, `BlockRequest` messages that are not answered in a timely fashion need to be sent to another peer in D_r. While a short timeout value causes the number of duplicate blocks received to increase, wasting resources, a longer time out makes the system react too slowly, increasing the number of blocks that do not meet their playback deadline. LiveShift tackles this problem by keeping a moving average of response time of each peer in D_r. When a requested block takes longer than twice the moving average of the last five block requests, the block is rescheduled. A default value of four seconds is used for the first block download attempt, since the average is not yet known. The value chosen shall include the time required for the `BlockRequest` message to be sent, processed, and replied to via a `BlockReply`.

3.4.3 Candidate and Neighbor Selection

Initially, peer r retrieves 40 random peers from the DT to be added to C_r. Peer r may also receive candidates from a peer p via `PeerSuggestion` messages, which should be sent following `DisconnectDown` messages, containing all known peers that are providers for the segment r was subscribed. This is important to quickly repair the mesh, causing less disruption. Furthermore, peers add senders of `Subscribe` messages for interesting segments as candidates, since these may be newly arrived peers.

Every peer tries to have up to 15 other peers in N_r by choosing from C_r in the following order: (1) least amount of `Subscribe` messages sent, (2) highest amount of blocks provided, (3) random. This selects peers that provided successfully blocks in the past, while allowing for rotation in case downloading is not successful.

Peers stop looking for members of C_r, N_r, and I_r when receiving video blocks at a rate sufficient to keep up with normal playback, that is, at least one block per second. This is to reduce overhead and avoid needlessly preempting other peers.

As a peer advances its playback position, it eventually reaches the segment boundary and needs to start downloading the next segment. For the new segment, a peer r adds to C_r and first tries to obtain upload slots at the peers from which it has successfully downloaded blocks in the recent past. This results in a more stable segment transition, since the peer does not have to begin again looking for peers to reach D_r.

Peers are removed from C_r after exceeding a threshold (currently 5) in number of `Subscribe` messages sent without having provided any blocks. This allows new peers to be added to C_r from the DT. Peers in N_r that report to only hold blocks too far (currently 8 s) behind playback position are removed from N_r, since they are unlikely to have interesting blocks soon.

3.4.4 Subscribers and Upload Slot Selection

It is intuitive that peers should prioritize uploading to peers with a high upload capacity, because these are able to serve many other peers, amplifying the upload capacity of the peercaster earlier in the distribution process [94]. By doing so, the average application-level hop count to the peercaster through the overlay is reduced, thus reducing average playback lag.

To this end, members of S_p and U_p are primarily chosen by peer p according to their upload capacity. $\overline{S_p}$ is defined as 5 peers per upload slot, that is, $5 \cdot \overline{U_p}$. This value should not be too large, since it is only worthwhile to keep subscribed peers that are likely to get an upload slot. In case there are peers with the same upload capacity, the peers which have been given more blocks in the recent past have preference, which increases overlay stability, avoiding unnecessary changes to the overlay. If there are still ties, a random peer is selected. A peer may only grant a single upload slot to a peer, in order to distribute streams to more peers.

Using a fixed number of upload slots has disadvantages. Possibly, the more upload slots a downloading peer is granted, the less often it will re-

quest blocks from each of them. It is difficult for an uploading peer to know precisely how much its upload slots will be used at any moment. The solution is having peers dynamically adjusting the number of upload slots according to the used upstream bandwidth. When a peer p detects that its upstream is underused by the granted upload slots, it creates a new upload slot and grants it to a peer in Q_p. When, however, the used upstream bandwidth reaches the maximum capacity of the peer and each slot is providing, on average, less than the full stream, a slot is shut down by sending a `Queued` message to the peer to which it is granted. By adjusting $\overline{U_p}$ to be able to provide at least the full stream at full rate to each slot, it avoids many peers receiving the stream at a rate too low to be played properly.

When selecting a slot to shut down, an intuitive policy would be using the same ranking used to grant upload slots. But since the last granted slot is possibly the lowest-ranked one, the system could return to the previous state of underusing upstream. Thus, the method currently used is to shut down the least used slot in a moving average of 3 s.

Concerning timeout values, T_S is set to 5 s, and T_Q to 10 s. T_I defines that a peer may remain only up to 4 s inactive (not downloading blocks) while granted an upload slot. Such low values are to promote rotation.

3.4.5 Playback Policy

The *playback position* is the position on the time scale that refers to the block currently being played, and is advanced by 1 block per second if the peer holds the corresponding block. Due to overlay network problems such as jitter, churn, upstream boundaries, and the limited view of resources in other peers, some blocks may not be found, or they may not be downloaded on time to be played. The *playback policy* is the decision on, when a block is missing for playback, whether to *skip* it, or *stall* waiting for it. While skipping has a negative effect on image quality, stalling increases playback lag.

LiveShift's playback policy is to skip x contiguous missing blocks if and only if the peer holds at least $x \cdot n$ contiguous blocks immediately afterward. A ratio $n = 2$ is used on the evaluations; in practice, though, it may be more adequate to let the user adjust the ratio to express its preference on whether

to skip or stall more often. The proposed policy and ratio are nevertheless effective at not letting peers stall for long in case only a few blocks are rare, achieving a low number of skipped blocks, which cause severe image quality degradation with most video encoding methods. An in-depth analysis of different playback policies is found in Chapter 4.

3.4.6 Storage Policy

The selection of which blocks peers keep in the local storage in case they run out of space is an interesting aspect and impacts data availability. The currently used storage policy is storing all received blocks until the maximum capacity – currently two hours of video – is reached. When storage capacity is full, a least recently used (LRU) policy is in place – blocks with oldest download time are deleted to make up space. Although simple, the strategy achieves good results, since blocks that are currently more popular naturally get more replicated in the system. The peercaster may choose to offer larger storage capacity in order to assure that at least one copy of past video streams is accessible in the system, though this is not assumed on this chapter's evaluation section.

3.5 Evaluation

LiveShift has been evaluated with the ultimate goal of validating its protocol and default policies in a realistic environment, by examining objective QoE metrics, as discussed in Section 2.4, namely playback lag and share of skipped blocks. The evaluation takes into account that P2P environments are heterogeneous regarding peer upload capacities [54], that peers tend to switch channels with a high frequency [18], and that peers join and leave the network (churn) [98]. Therefore, the system has been fully implemented, in Java version 1.6, allowing an insight into the running protocol.

To improve experiment reproducibility, a research testbed was used. The evaluation was made using 16 physical machines from the Communication Systems Group (CSG) testbed – a detailed description of the testbed can be found at [113]. Though machines were diverse, a monitoring process was

in place to assure that CPU and memory limits were not reached, which would bias results. Thus, the physical bottlenecks assumed are the network latency and bandwidth limitations introduced.

This section presents in detail the evaluation environment, as well as the subset of obtained evaluation results which pertains to the validation of the proposed protocol and policies, the testing of its scalability limits, and the investigation of its improvement opportunities. Evaluation includes both channel browsing behavior and churn to produce highly realistic results. The reader is directed to [45] for additional evaluation results.

3.5.1 Evaluation Scenarios and Peer Behavior

Table 3.3 describes the four scenarios addressed. These were selected to be heterogeneous regarding peer capacities, to show the system's ability to support several peers with upload capacity lower than the bitrate of the video stream being transmitted. Thus, peers were divided in classes regarding their maximum upload capacities. High upload capacity (HU) peers and peercasters (PC) have upload limit of 500 percent, e.g., for the video stream bit rate of 500 kbit/s defined for these experiments, HU and PC nodes have an upload capacity of 2.5 Mbit/s. This value is not particularly high, considering that these nodes may be running at universities or connected via FTTH (Fiber to the Home) technology. Low upload capacity (LU) peers may be, for example, running DSL (Digital Subscriber Line) or cable connections, and have only 50 percent upload capacity (250 kbit/s).

The total upload capacity is calculated by adding the upload capacity of every peer in the scenario. Scenarios were carefully chosen such that the scenario load (total peers divided by total upload capacity) is increasingly high, since this is a very challenging situation for P2P streaming systems. Such scenario choices, with an increasing number of LU peers, allow the LiveShift system to be evaluated under different levels of scenario load. While Scenario s1 has a low (60%) Scenario Load, meaning that there is spare upload bandwidth available in the system, the overloaded Scenario s3 has 84% of its total upload capacity compromised.

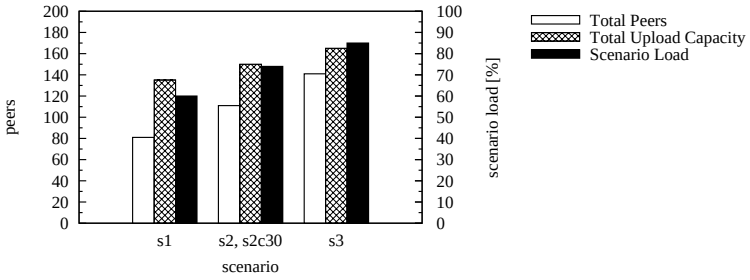

Figure 3.5: Evaluation scenarios capacities

Figure 3.5 further illustrates the increasing scenario load. The total upload capacity does not increase as much as the number of peers from Scenario s1 to s3. The scenario load, hence, becomes higher.

Finally, peers are not limited in download bandwidth, but testbed nodes are connected via 100 Mbit/s links. All results were obtained over 10 runs of 1 hour each. A warm-up phase is not omitted from the results, since it creates very low traffic.

Table 3.3: Evaluation scenarios

Scenario	*PC Peers*	*HU Peers*	*LU Peers*	*Churn*	*Total Peers*	*Total Upload Capacity*	*Scenario Load*
s1	6	15	60	0	81	135	60%
s2	6	15	90	0	111	150	74%
s3	6	15	120	0	141	165	85%
s2c30	6	15	90	30%	111*	150*	74%*

* these are maximum values; actual values are influenced by churn

Peer behavior was modeled using traces obtained from a real IPTV system [18]. Peers were created with an inter-arrival time of 1 s and loop through the following two steps: (a) choose a channel and starting time, (b) hold to the channel, locating and downloading content from other

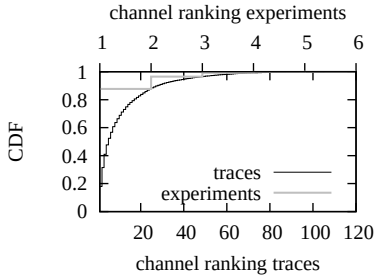

Figure 3.6: Channel popularity

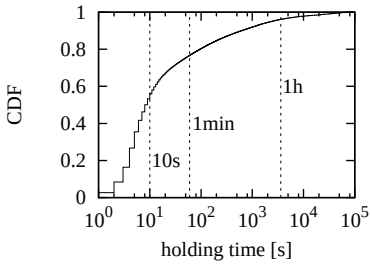

Figure 3.7: Channel holding time

peers. The distribution used to determine which channel peers switch to is displayed in Figure 3.6, including the original distribution with 120 channels, and the one used on these experiments with six channels. The channel holding time distribution is pictured in Figure 3.7. Since there is no system supporting both live and time-shifted streaming as described on this chapter, there are no traces available documenting how long behind real time will a newly arriving peer join. Hence, it was assumed that the probability of a peer tuning to a time nearer the current *live* time is exponentially larger than the probability of it tuning to some other point in the past. The distribution used was the one pictured in Figure 3.7, with the starting time ranging from the current playback position, and all the way back to the playback time at the start of the experiment. Note that the traces for holding time distribution include many short (55.97% are less than 10 seconds long) sessions, as well as some long ones (3.89% are over one hour).

In the scenario with churn (s2c30), when a peer chooses a channel and time to tune to, it has a nonzero probability of going offline. While offline, a peer does not react to any incoming message. Peers remain offline for an amount of time given by the channel holding time distribution before having again the same chance of remaining offline or going back on-line. Peers disconnect cleanly, that is, they follow the protocol properly by sending `PeerSuggestion`, `DisconnectUp`, and `DisconnectDown` messages.

3.5.2 Playback Lag

The main QoE metrics used are the playback lag experienced and reported by the peers as the playback position advances, from the point a (*channelId*, *startTime*) was selected, as well as the share of skipped blocks. An in-depth study of playback and the definition of playback lag are given in Section 4.1. Reports from each peer and run were collected and an average was calculated for every 1-minute interval for clarity of display. The same proceeding was performed on all runs.

Figures 3.8-3.11 show the playback lag experienced by users at a watching session in the different proposed scenarios. The x-axis represents the playback position (in minutes), as the sessions advances. The y-axis represents the playback lag (in seconds), which increases when playback stalls, but decreases when blocks are skipped – skipped blocks are shown in Section 3.5.3. A lower playback lag means lower start-up delay, less stalling, and more closeness to what the user initially intended to watch. Each line represents a p-percentile of peers (HU or LU) over all channels and all runs. Because sessions are of various lengths, according the holding time distribution shown in Figure 3.7, the number of peers from which the p-percentile is taken is inversely proportional to the playback position.

For example, the 95% LU line designates the maximum playback lag which the 95 percentile of low upload capacity users achieve. In other words, it is the worst case lag for 95 percent of LU peers. Since blocks in LiveShift are transmitted via reliable connections (TCP), there is no risk of losing data within a block.

As can be seen in Figures 3.8-3.11, results show that:

a. playback lag increases only slightly as users continue to view a channel, which shows that users do not experience long stalling events;

b. even in the worst case scenarios investigated, 95 percent of HU peers experience playback lag of less than 10 seconds, which is acceptable performance (indeed many live TV broadcasts may have similar lag); and

c. LU peers are much more susceptible to high lag and especially in scenarios with churn or less available bandwidth. For instance, Figure 3.11

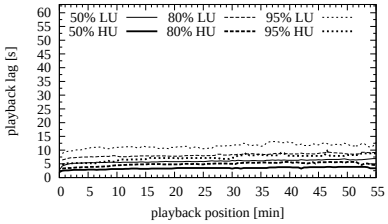

Figure 3.8: Playback lag in s1

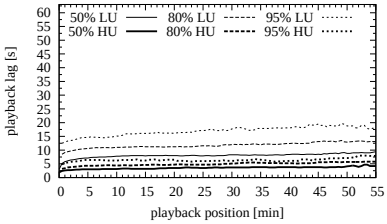

Figure 3.9: Playback lag in s2

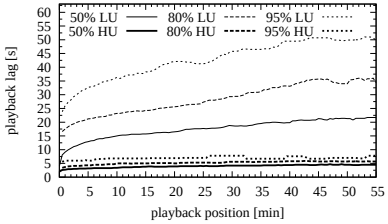

Figure 3.10: Playback lag in s3

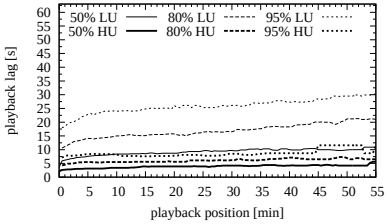

Figure 3.11: Playback lag in s2c30

shows that the 5th percentile of LU peers exhibit a playback lag larger than 25 seconds after watching for long periods of time, while in Figure 3.10, the worst 5 percent of LU peers have playback lag just above 50 seconds after watching a channel for more than 40 minutes.

In Scenario s3, the system shows signs of being saturated, with several LU peers exhibiting playback lags surpassing 30 s. This happens because the average delay to the peercaster through the overlay increases, since the blocks need to travel through relatively more hops. In addition, peers take a longer than average time to obtain upload slots, which are more disputed in this scenario.

Overall, average playback lag is 5.45 s in s1, 7.70 s in s2, 14.31 s in s3, and 8.93 s in s2c30, showing that 30 percent of churn increases average playback lag by about 15 percent in s2.

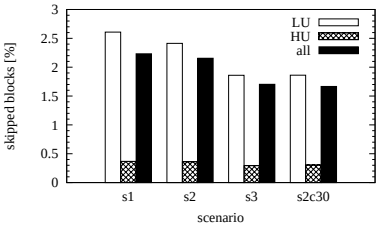

Figure 3.12: Skipped blocks

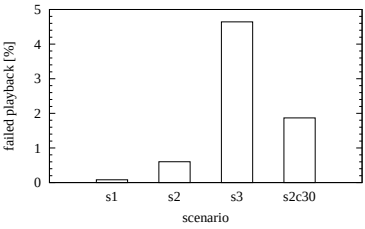

Figure 3.13: Failed playback

3.5.3 Skipped Blocks and Failed Playback

According to the playback policy defined in Section 3.4.5, some blocks may be skipped, therefore causing a decrease in playback lag. It can be seen in Figure 3.12 that the proportion of skipped blocks is nonetheless low, that is, lower than 3% in all investigated scenarios. Interestingly, relatively less blocks are skipped in more bandwidth-constrained scenarios, which is due to fewer concurrent downloads. HU peers skip very few blocks, since they are closer to the peercaster, thus enjoying less instability. An in-depth study of playback policies and their effects in playback lag and share of skipped blocks is presented in Chapter 4.

The availability of content is affected by the fact that peers change their interest frequently. In the worst case, a peer may not be granted an upload slot from any of the peers which hold the blocks that it seeks. This may happen even when the system has spare bandwidth, due to the unbalance in content popularity: peers that have unused upload capacity may only hold unpopular content, leading to available overlay resources remaining unused. If playback stalls for a long time, it is not realistic to assume that the user will wait forever. Thus, when a peer, in a sliding window of the last 30 s of playback, is able to play less than half the blocks it should have been able to play, playback is considered failed, that is, the user is considered to have given up and switched to another $(channel, time)$. Failed playback events as percentage of channel switches can be visualized in Figure 3.13.

3.5.4 Upload Capacity Utilization

The upload capacity utilization of a peer is its percentage of upload capacity used, on average, and is obtained by dividing its *used* upload capacity by its *total* upload capacity. An efficient P2P system is able to discover unused bandwidth and react quickly to peers changing interest, which is challenging in a fully-decentralized system.

Figure 3.14 shows the average upload capacity utilization, per peer. The fact that HU peers use more of their upload capacity, is a side effect that they placed nearer the PCs, due to the policy in Section 3.4.4. Thus, they receive (and announce) having blocks sooner than LU peers. As the scenario load increases (from Scenarios s1 to s2 and s3), the upstream utilization of LU peers increases quickly. It can finally be confirmed that the system in scenario s3 is close to its limit concerning upload capacity, as both HU and LU peers upload above 80% of their capacity on average.

There is little variation in upload capacity utilization of peercasters in the different scenarios because they do not react to channel popularity – in s2, while the PC for channels 1 and 2 average higher than 98 percent upload capacity utilization, the PC for channel 6 averages only 4.0 percent simply because the channel is unpopular.

In addition, due to the difference in popularity of channels and the dynamic behavior of peers, some swarms may be more or less provided with bandwidth. This can explain why some LU capacity is used while HUs are not fully loaded: some swarms may not have enough HUs, and need to resort to LUs. Because of the small block size, peers can combine the upstream capacity of several LU peers.

3.5.5 Overhead

Since both blocks and segments are fully time-based, the absolute system overhead does not depend on the bit rate of the video stream, but rather, on the length of blocks and segments.

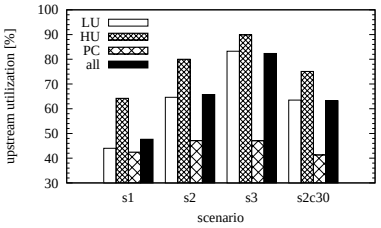

Figure 3.14: Upstream utilization

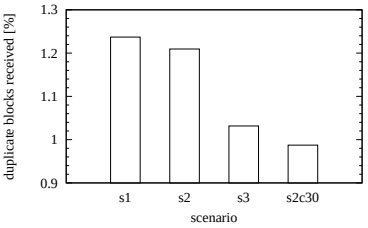

Figure 3.15: Duplicate blocks

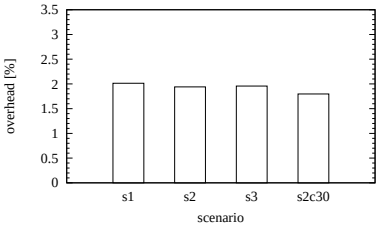

Figure 3.16: Overhead, not including DHT+DT

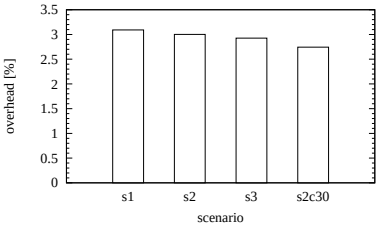

Figure 3.17: Overhead, including DHT+DT

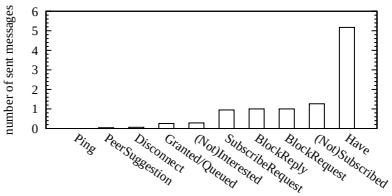

Figure 3.18: Sent messages per peer per second in Scenario s1

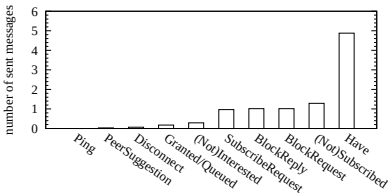

Figure 3.19: Sent messages per peer per second in Scenario s2

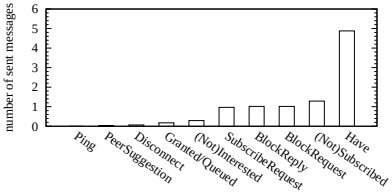

Figure 3.20: Sent messages per peer per second in Scenario s3

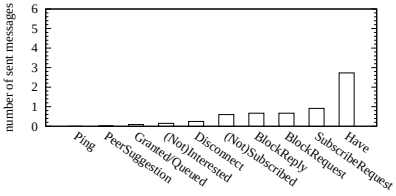

Figure 3.21: Sent messages per peer per second in Scenario s2c30

Figures 3.18-3.21 give an insight into the running protocol, displaying the average number of messages a peer sends per second, by type. Some messages `Have` messages are the most common ones, which is expected from the protocol design. The number of `Have` messages sent per second mean that, on average, $|S| = 4.88$ in s2. All other messages have negligible size when compared to the `BlockReply` message, which carries the video block itself.

The overhead relative to a stream of 500kbit/s is, for any scenario, at most 2.01 percent on average, excluding DT and DHT traffic, as can be seen in Figure 3.16. Figure 3.17 includes DT and DHT traffic, and shows that overhead is at most 3.09 percent on average. Interestingly, Scenario s1 has slightly higher overhead than the other scenarios, which is credited to the higher number of messages allowed by the higher available bandwidth.

Figure 3.15 shows that the proportion of duplicate blocks received is below 1.3 percent on average, for all the analyzed scenarios. Similarly to the number of skipped blocks, the decrease in more bandwidth-restricted scenarios is explained by a smaller $|D|$, which means a restricted choice of peers to download video chunks from. This validates the rescheduling policy presented in Section 3.4.2.

3.6 Chapter Summary

This chapter has introduced the LiveShift system, including the LiveShift application and the LiveShift protocol, and has identified a set of policies that alter the behavior of the protocol. The system has shown its ability of maintaining a low playback lag relatively to the playback position through realistic evaluations, by running a full implementation of the LiveShift application, and including both channel switching and time shifting at a high frequency, a high ratio of peers with insufficient upload capacity, and churn.

4

Playback Policies for Live and On-Demand P2P Video Streaming

This thesis identifies two related but orthogonal issues: the definition of protocol and policies for live and time-shifted multimedia streaming. The protocol definition, as introduced in Section 3.3, characterizes message formats and their semantics, representing a common language used by the diverse peers in the system. A good P2P protocol, though, is flexible, by allowing participants to adapt and optimize its usage according to the environment and their particular preference. Policies, thus, define how the protocol is used by the peers, for instance, in the case of LiveShift, how providers are selected and which peers to grant upload slots to.

As shown in Section 2.5, while some policies identified in Section 3.4 have obtained great attention from the research community, playback policies are greatly understudied. The *playback policy* is responsible for defining whether peers skip playing blocks that are unavailable, or stall playback, waiting for them to be found and downloaded.

The two main research questions this chapter addresses are (a) *do different playback policies affect user experience in a P2P video streaming system*, and (b) *which playback policies are most suitable for live and on-demand sce-*

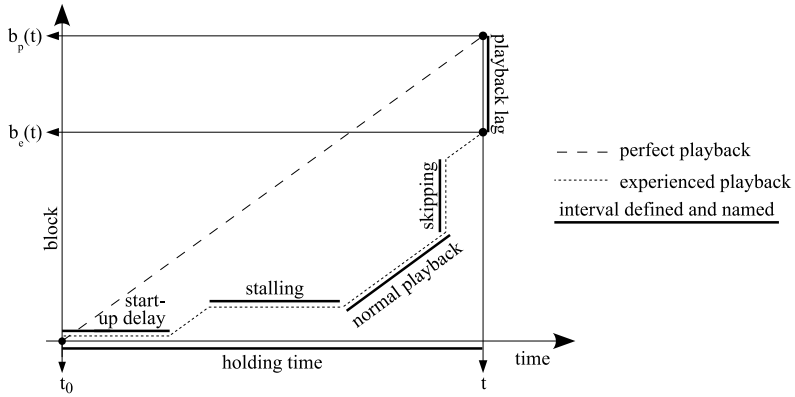

Figure 4.1: Playback terminology

narios? In order to answer these questions, this chapter briefly overviews P2P video streaming and introduces key terminology in Section 4.1. A classification and generalization of different playback policies are presented in Section 4.2, enabling a meaningful comparison among them. These policies have been implemented in LiveShift; Section 4.3 discusses the evaluation and comparison of those under a variety of carefully selected scenarios and parameters. Since LiveShift enables an evaluation of both live and on-demand scenarios, those results obtained are generalizable. Finally, Section 4.4 summarizes this chapter's main contributions.

4.1 Background and Terminology

LiveShift, like most successfully-deployed P2P video streaming systems, employs the mesh-pull approach [53]. Mesh-pull consists on dividing the stream into *blocks* that are announced and exchanged directly between pairs of peers with no fixed structure. Peers read obtained blocks from a buffer (or a long-term storage in case of LiveShift, to support time shifting), sending them to a video decoder that displays the content to the user. The playback policy is implemented at the component that selects which blocks are to be played, that is, the Player Sender from Section 3.2.

Table 4.1: Playback nomenclature

L	Block length (time unit)
t_o	Session start time
t	Current playback position
$b_p(t)$	Block played at t if perfect playback
$b_e(t)$	Block played at t
$t_{lag}(t)$	Playback lag at t
$t_{st}(t)$	Time stalled from t_o until t
$n_{sk}(t)$	Skipped blocks from t_o to t
$n_{pl}(t)$	Played blocks from t_o to t
ℓ	Buffer size (in blocks)
α	Initial buffering coefficient
β	Stalling coefficient
t_d	Remaining download time
t_p	Movie length (time unit)
r	Relative incoming block rate
T	Maximum retries
n	Minimum block ratio

Figure 4.1 illustrates various terms used in this chapter, and Table 4.1 defines nomenclature. In LiveShift, blocks have a fixed length L in the time scale. A viewing *session* starts when a user chooses a channel and starting time t_o (for live streaming, t_o is the current time). While the user holds on to (i.e. watches) a channel, the system attempts to locate and download the corresponding blocks. Ideally, the user would experience *perfect playback*, that is, without interruptions, the block to be played $b_p(t)$ would be obtained based on the *playback position* t:

$$b_p(t) = t/L \tag{4.1}$$

However, due to lack of content availability that leads to stalling and skipping, experienced playback is not always normal (represented by diag-

onal lines in Figure 4.1). At time t, a user actually watches block $b_e(t)$, as given by (4.2), where $n_{sk}(t)$ is the number of skipped blocks and $t_{st}(t)$ is the time stalled from t_0 until t.

$$b_e(t) = n_{sk}(t) + (t - t_{st}(t))/L \tag{4.2}$$

Performing *initial buffering* corresponds to stalling playback until the playback buffer accumulates a number of blocks (typically a fixed configuration parameter), as an attempt to reduce the chance of skipping or stalling during playback. The *start-up delay* is the experienced stalling time caused by initial buffering. The *playback deadline* determines the time that a particular block is due to be played, according to the playback policy. Start-up delays are accepted by the users to avoid stalling or skipping [55].

The term playback lag is commonly defined for live streaming as the elapsed time from the moment a block is generated at the source until it is played at the peer side [72]. In LiveShift, the concept of *playback lag* is extended also for viewing time-shifted streams; playback lag is thus defined by

$$t_{lag}(t) = (b_p(t) - b_e(t)) \cdot L \tag{4.3}$$

as the time difference between the block that is playing, according to the playback policy used, and the block that would be playing, if there were no interruptions from the moment the user pressed the play button. This extension in the definition of playback lag preserves its original concept of measuring the overall ability of the system of locating and downloading content, while being applicable to non-live viewing as well. *Liveness* is a term used in the literature that stands for low playback lag, though it is not usually associated with a unit; for example, a system that loses liveness with longer sessions suffers an increase in its playback lag.

While both stalling and skipping negatively impact user-perceived video quality, their impact is different. On one hand, when stalling occurs, the video stream is interrupted and playback lag is increased. Besides, peers with higher playback lag lose their ability of providing streams with more liveness to other peers, negatively impacting the entire system. On the other hand, when skipping occurs, image quality might be impaired, since

part of the video data is missing. Besides, since skipped blocks are not downloaded, they cannot be uploaded to any other peer, creating buffer "holes" that may harm the distribution overlay. In both cases, peers need a larger number of providers to compensate for those missing blocks, which may be challenging, since upload capacity is typically a rare resource in such systems [53].

4.2 Playback Policies

This section describes and generalizes a set of four playback policies, based on the extensive related work survey presented in Section 2.5, plus a new Catchup policy, to enable a fair and meaningful comparison among them. The four playback policies based on related work represent the current state-of-the-art in the field. Analysis and discussion on respective trade-offs in both live and on-demand scenarios are, as well, performed.

The Always Skip policy, commonly used for live streaming, consists on always skipping missing blocks to maintain liveness. It is defined by $P_{as} = (\ell, a)$, where ℓ represents the size of the playback buffer (in blocks), and $a \in (0, 1]$ corresponds to the share of blocks that the buffer must hold before starting playback. After the first block has been played, buffered blocks are attempted to be played sequentially and at constant speed. Missing blocks are immediately skipped; however, if the buffer is empty, playback stalls to perform again initial buffering. This is done so peers adapt their playback position $b_e(t)$ according to the blocks that can be found at – and downloaded from – currently known peers.

The Skip/Stall (`sk`) policy is an extension to the Always Skip policy to allow stalling as in Tribler [86]. It is defined as $P_{sk} = (\ell, a, \beta)$, which introduces the $\beta \in [0, 1]$ coefficient, such that, when a block at $b_e(t)$ is missing and the buffer is not empty, the system stalls playback until a share β of the playback buffer is filled; then, it skips to the next available block. The Always Skip policy is, thus, an instance of the `sk` policy when $\beta = 0$.

Especially for VoD, it is reasonable to define a playback policy that depends on the remaining download time, so stalling is reduced for users with a fast network, while buffering increases with slower networks. The

Remaining Download Time (rd) policy elaborates on related work such as [87], that describe stalling playback until the remaining download time $t_d \leq t_p$, the remaining playback time, e.g., movie length, defining it in more precise terms.

In order to apply this policy to LiveShift, the concept of remaining playback time must be defined, since the stream ahead of playback position may be very large – it extends until the current (live) time. Hence, t_p is a parameter that may be set to, e.g., 30 minutes of video that buffering will attempt to guarantee playback for.

The rd policy can be modeled as $P_{rd} = (\ell, a, \beta, t_p)$ by using the same algorithm as defined for the sk policy, but using a variable buffer size ℓ', calculated based on the parameters t_p and ℓ, instead of ℓ directly. Though not described in [87], infinite geometric series are useful to calculate how long the playback buffer would last, since the application continues to download blocks while buffered blocks are played back. If i represents the incoming block (*i.e.* download) rate, and L being block length, let r represent the relative incoming block rate, such that $r = i \cdot L$; thus, if $r = 1$, the peer is downloading blocks at a rate exactly enough to keep normal playback. The variable buffer size ℓ' can therefore be calculated as shown in Equation 4.4, where ℓ is used both for initial buffering and as a general lower limit of the buffer size (when, for example, $r \geq 1$). The coefficient a is present in the equation to preserve the semantic of remaining download time, since only a share a of the buffer is required by the playback policy to be held in the buffer.

$$\ell' = \max\left(\frac{t_p}{L} \cdot \frac{(1-r)}{a}, \ell\right) \tag{4.4}$$

The Retry (re) playback policy is similar to the policy implemented in Gridcast [20], and is defined as $P_{re} = (\ell, a, T)$. It consists on performing initial buffering, then stalling if the block at playback position t is not available. The system retries playing the missing block up to T times, which brings playback to stall for a maximum of $T \cdot L$ seconds. As soon as the missing block is downloaded, it will be played back; if the stalling threshold is hit, though, playback skips to the next available block.

The Ratio (`ra`) policy aims at skipping blocks only if there is a high chance of then continuing playback without interruptions, as described in Section 3.4.5. It is formally described as $P_{ra} = (\ell, a, n)$, where ℓ and a retain their previous meaning. After initial buffering, if the block at playback position is locally held, it is always played. If, however, the block is missing, a ratio $1 : n$ is given, such that x contiguous missing blocks are skipped, if and only if, at least $x \cdot n$ contiguous blocks are held directly after those.

The Catchup (`ca`) playback policy is introduced to keep playback lag very low at the cost of skipping more blocks than other policies. It is defined by $P_{ca} = (\ell, a)$, where ℓ and a are used to perform initial buffering as in the `sk` policy. After playback has started, all missing blocks are skipped, as long as the buffer is not empty. When it is indeed empty, playback position is restored to the original one by skipping all blocks responsible for playback lag until $b_e(t) = b_p(t)$. It is meant to provide a practical limit on the lowest possible playback lag achievable.

4.3 Evaluation

All playback policies defined in Section 4.2 have been implemented into LiveShift. Implementation details are described in Section 6.2, which show their integration with the rest of the LiveShift system. Experiments were conducted using the entire LiveShift code.

The main objective of the evaluation is to compare how different playback policies affect user experience of a P2P video streaming system under scenarios with different levels of content availability. This is achieved by running multiple instances of LiveShift under realistic but comparable assumptions, in terms of bandwidth and latency limitations, channel popularity, channel switching, time shifting, and scenarios used.

Table 4.2 displays those different scenarios used. Similarly to Section 3.5, peers are divided into classes according to their maximum upload capacities – while high upload capacity (HU) peers and peercasters (PC) are able to upload at a rate equivalent to 5 times the bit rate of the video stream being transmitted, low upload capacity (LU) peers are able to upload at only 0.5 times the original stream rate. The increasing number of LU

peers causes available upload bandwidth to decrease; while Scenario s1 has abundance of total upload capacity compared to the number of peers to be served (thus, a low Scenario Load), in Scenario s4, the chance that peers experience content unavailability is much higher (Scenario Load is very high). It is important to note that peers that have unused upload capacity might only hold unpopular content, leading to suboptimal overlay resource usage. Thus, the scenarios chosen for evaluating the defined playback policies represent a wide variety of situations, with different levels of content availability. Peers are not artificially limited in download bandwidth, and latency between peers was introduced by using a random sample from the King dataset [41], and enforced using DummyNet [1]; the sample contains an average latency of 114.2 ms.

It is noted that, in contrast to Section 3.5, a more-heavily loaded scenario s4 has been added, due to the fact that playback policies are particularly suited for highly-loaded scenarios, as will be seen in the evaluations results. Evaluation scenarios with churn in terms of peers disconnecting and reconnecting to the network has been, in scope of playback policies, left for future work. However, churn has been considered in terms of peers switching channels.

The same test-bed as in Section 3.5, but with 20 machines [113], was used to run all experiments. Though machines were diverse in capacity, they were monitored to guarantee that CPU and memory limits were not reached. Thus, the main bottlenecks are the network latency and bandwidth limitations introduced.

Table 4.2: Playback policies evaluation scenarios

Scenario	*PC Peers*	*HU Peers*	*LU Peers*	*Total Peers*	*Total Upload Capacity*	*Scenario Load*
s1	6	15	60	81	135	60%
s2	6	15	90	111	150	74%
s3	6	15	120	141	165	85%
s4	6	15	150	171	180	95%

Multiple instances of LiveShift were executed, adopting the same behavior as described in Section 3.5. Peers were created with an inter-arrival time of 1s. Every peer was programmed to repeatedly switch to a channel and starting time t_0, then hold to the channel, attempting to locate and download blocks. While holding to the channel, every peer reported, once per second, its experienced playback lag $t_{lag}(t)$, as defined in Section 4.1, and share of skipped blocks $n_{sk}(t)/(n_{pl}(t) + n_{sk}(t))$. Channel popularity and holding time were both characterized by traces, as described in Subsection 3.5.1 and depicted in Figures 3.6 and 3.7. Results were obtained through 10 runs of 20 minutes each; 10 runs due to the observation that new runs were producing results similar to the ones already obtained, and 20 minutes being enough to observe the increase in playback lag.

Due to severe content unavailability, a peer may sometimes experience very long stalling times. In such cases, it is not realistic to assume that users would wait indefinitely for the content. Thus, when a peer is able to play less than 50 percent of due blocks in a moving window of 30 seconds, playback is considered *failed*, that is, the user is considered to have given up and switched to another $(channel, t_0)$. Buffering is not taken into account, since it is part of the playback policy being investigated. This is done to avoid peers being stalled for a very long time, and failed playback sessions are reported separately.

Since the goal of the evaluation is to perform an overall comparison of playback policies in the entire overlay, on each run, all peers were configured to adopt one of the playback policies defined in Section 4.2. Each playback policy has been investigated using values as specified in Table 4.3 for their main parameters, which are based on quantities seen in the literature, complemented by additional values that allow a deeper understanding of their effect. Hence, the Skip/Stall policy was experimented with parameter $\beta = 0$ to yield an always skip policy as measured in proprietary live streaming systems [103], $\beta = .5$ to match one of the Tribler policies [86], and $\beta = .75$ as an additional value. The Remaining Download Time policy was set with parameters $t_p = 5s, 30s, 60s$ to include both short and larger time spans. Parameter values used with the Retry policy were $T = 10$ as in [20], plus $T = 5$ and $T = 1$, since retrying playing a block 10 times on a video

Table 4.3: Playback policies and parameters

Policy	*Parameter*	*Identifier*
Skip/Stall	$\beta = 0$	`sk-0`
	$\beta = .5$	`sk-.5`
	$\beta = .75$	`sk-.75`
Remaining Download Time	$t_p = 5s$	`rd-5`
	$t_p = 30s$	`rd-30`
	$t_p = 60s$	`rd-60`
Retry	$T = 1$	`re-1`
	$T = 5$	`re-5`
	$T = 10$	`re-10`
Ratio	$n = 2$	`ra-2`
	$n = 3$	`ra-3`
	$n = 5$	`ra-5`
Catchup	(none)	`ca`

streaming system seemed rather large. For the Ratio policy, experiments involved $n = 2$ as in Section 3.4.5, complemented by slightly larger ratio values $n = 3$ and $n = 5$, since the n value alters largely the number of blocks required to be held in the buffer in order to skip playback. To make results better comparable, parameters that apply to all playback policies were kept constant; all experiments were obtained using $\ell = 6s$, $a = 0.8$, and $L = 1s$. The evaluation metrics used are playback lag, share of skipped blocks, and share of failed playback sessions.

4.3.1 Playback Lag

Playback lag is directly related to user experience, since a lower value denotes a lower start-up delay, less interruptions, and more closeness to what the user initially intended to watch. Playback lag is expected to increase with larger sessions, as well as with lower content availability, due to stalling. Reports from each peer and run were collected and an average

was calculated for every 1-minute interval for clarity of display. The same proceeding was performed on all runs.

The distribution of playback lag among different peers and at different t values was analyzed for each policy. For most peers, playback lag differences for the investigated parameters are consistent, as exemplified in Figure 4.2, which shows the CDF of playback lag at 10 min playback position under the sk policy. Peers, however, with highest playback lag (*i.e.* above 90th percentile) suffer from severe content unavailability, as a result of the high channel switching frequency in the defined peer behavior, combined with the upload bandwidth constraints; these peers are not able to download any blocks, so the playback policy cannot play a significant role.

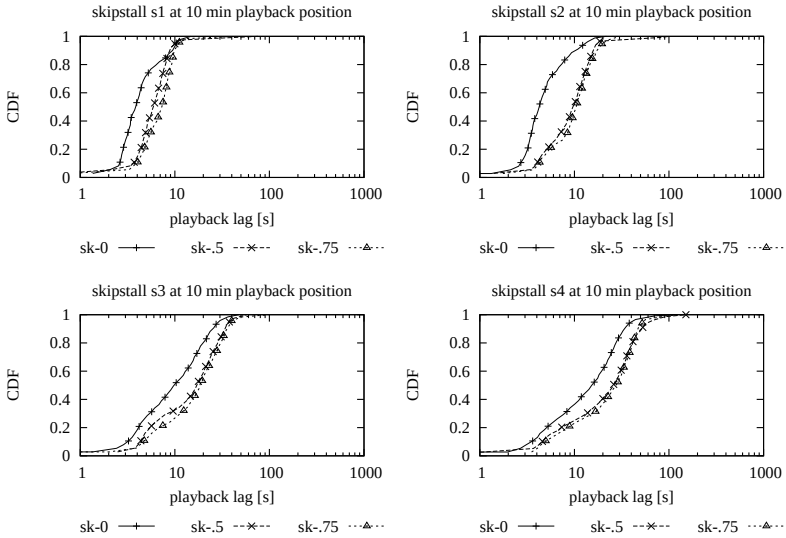

Figure 4.2: CDF of playback lag under Skip/Stall playback policy

Since this occurs as well under all other investigated policies (*c.f.* Appendix B), all other playback lag plots in the rest of this section display, for clarity, only the 80th percentile, that is, the maximum playback lag experienced by 80 percent of the peers.

Experiments with the `sk` playback policy were made using three different values for the parameter β, as shown in Figure 4.3. While the x-axis represents the time t, in minutes, for which a user holds on to a channel, the y-axis represents the playback lag $t_{lag}(t)$, in seconds.

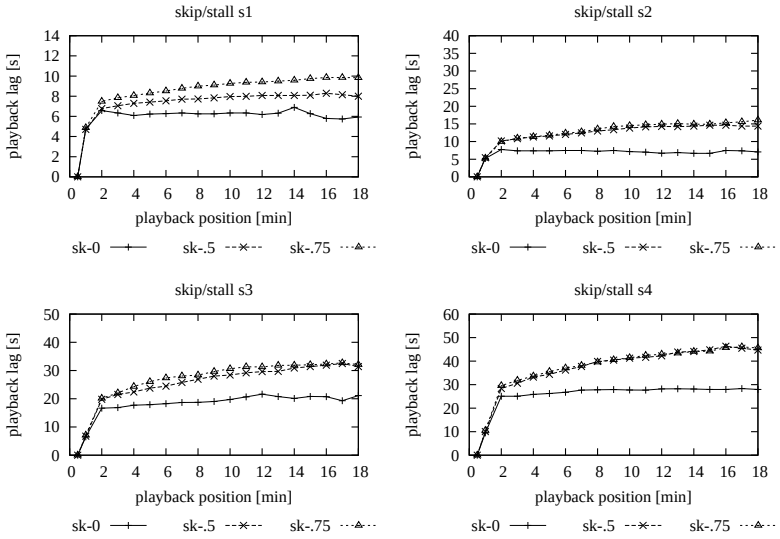

Figure 4.3: Always Skip and Skip/Stall playback policies

Evaluation results reveal that the `sk` playback policy is extremely flexible; depending on the value used for the parameter β, a wide range of resulting values for playback lag are experienced. It is able to maintain a relatively low playback lag even for longer sessions (higher playback position values) when $\beta = 0$ (`sk-0`, the Always Skip policy). In Scenario s1, `sk-.5` and `sk-.75` display very distinct results, yet on the other scenarios, they yield very similar results in terms of playback lag. This is due to the fact that, in a scenario with more available upload bandwidth, peers have more opportunity to perform several parallel downloads, hence the chance that a peer is able to download blocks out of order (thus being able to skip) is higher.

It can be observed under the Skip/Stall policy, as well as will be seen under the remaining playback policies, that playback lag increases at a much higher rate until playback position reaches about two minutes time. During this initial phase, peers attempt to find stable peers to download blocks from; however, this task is undermined by the channel switching behavior of peers, increasing playback lag.

Remaining Download Time Playback Policy

The `rd` playback policy was instantiated with different values of t_p, which is the minimum remaining playback time the policy attempts to guarantee playback for, considering the current download rate. Figure 4.4 shows that, on the over-provisioned Scenario s1, results differ little with the different parameters evaluated. This is due to the fact that peers can often download at a rate $r \geq 1$, therefore ℓ' frequently reaches its minimum value ℓ, as ℓ' decreases with a higher download rate. In scenarios s2 and s3, varying the parameter t_p yields very distinct results, and playback lag increases significantly with upload bandwidth scarceness. In the most bandwidth-restricted Scenario s4, larger values of t_p cause higher playback lag with higher playback positions, as expected, since the download rate r is lower. In comparison with other playback policies, the Remaining Download Time playback policy shows the highest playback lag, which is due to the potentially larger buffer size, especially with lower download rates.

Retry Playback Policy

The `re` playback policy was investigated with different values for the parameter T, which expresses the stalling limit per block. Figure 4.5 shows that, in all investigated scenarios, while re-1 displays a lower playback lag than `re-5` and `re-10` (as expected), it is still higher than levels achieved under the `sk` policy.

The fact that playback lag under `re-5` and `re-10` policies are very similar is due to the unlikelihood, in all scenarios, of situations in which playback needs to stall for longer than 5, but less than 10 seconds, until the block

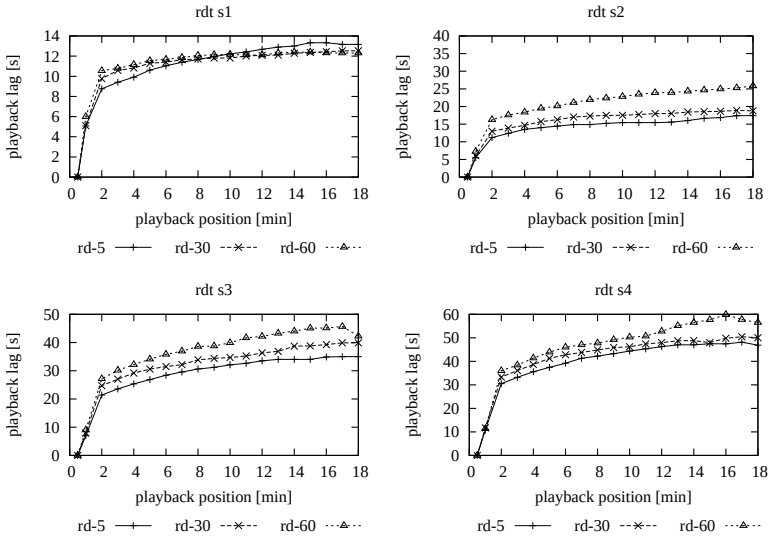

Figure 4.4: Remaining Download Time playback policy

at playback position is downloaded. Thus, a policy like Gridcast [20] of retrying 10 times each block causes high playback lag, and the effect would be similar even if a value of 5 was employed.

Ratio Playback Policy

Results with the `ra` playback policy were obtained using different values for the parameter *n*. Figure 4.6 shows that the different parameters used do impact experienced playback lag distinctively in the over-provisioned Scenario s1, but produce similar results in the other investigated scenarios. Similarly to the `sk` policy, peers have much fewer opportunities to skip blocks in s1 due to the lower probability of performing parallel downloads (higher chance that a single or very few peer have enough upload capacity to serve the whole stream). The fact that the Ratio playback policy requires blocks to be skipped to be contiguous makes skip situations rarer, hence the higher playback lag.

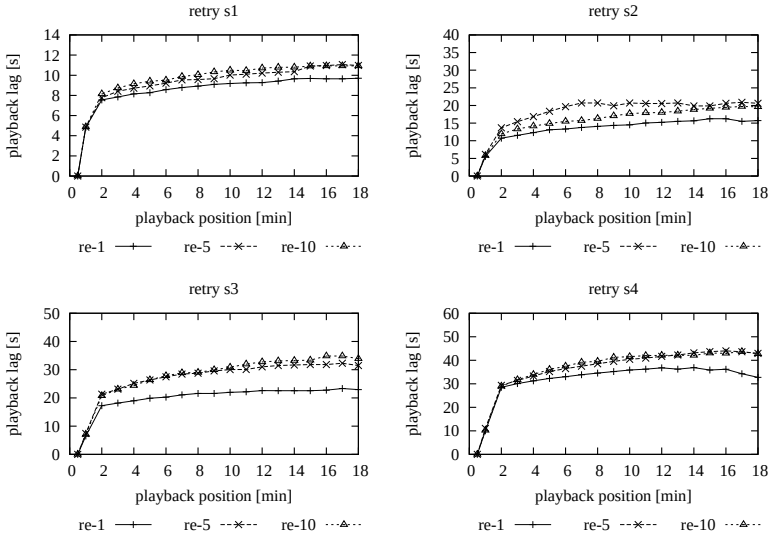

Figure 4.5: Retry playback policy

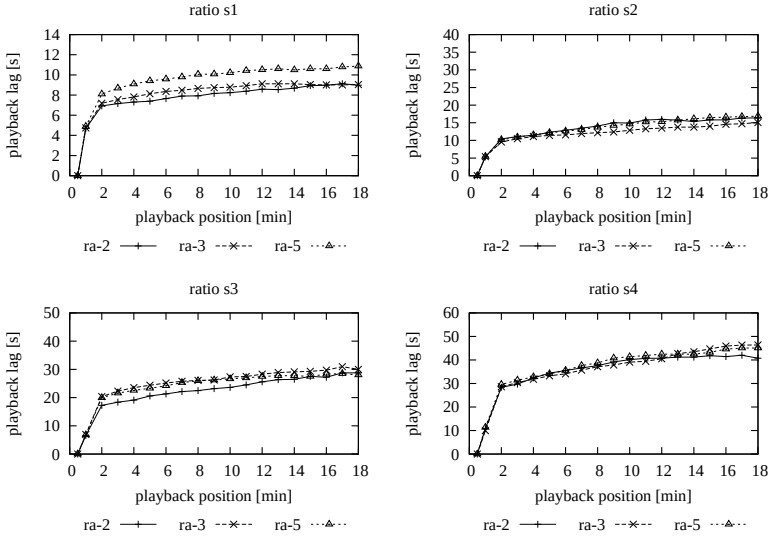

Figure 4.6: Ratio playback policy

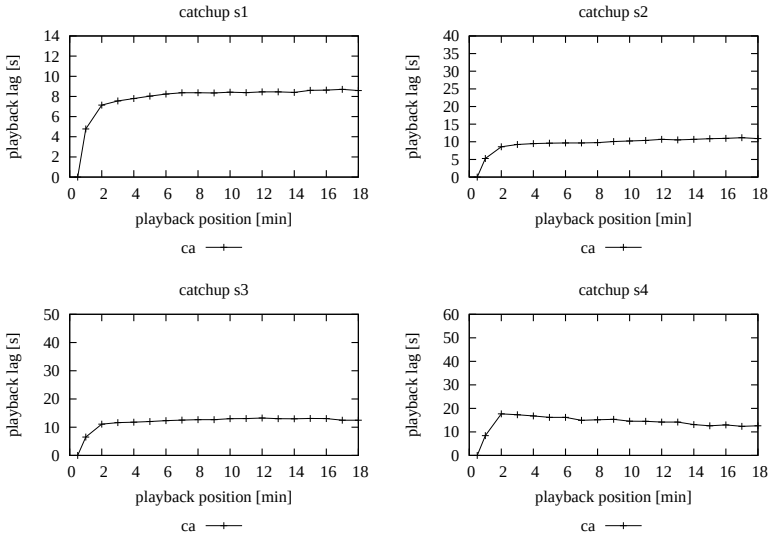

Figure 4.7: Catchup playback policy

Catchup Playback Policy

The Catchup (`ca`) playback policy is designed to keep a very low playback lag by resetting it to zero when the playback buffer is empty by skipping the necessary amount of blocks. Results show in Figure 4.7 that, as designed, it displays a relatively lower playback lag in comparison to the other policies, remaining stable as playback position advances. Interestingly, while it displays in Scenario s1 a clearly higher playback lag than `sk-0`, the opposite is observed in s4. This happens due to the much higher probability in Scenario s4 that the buffer becomes empty and the catchup mechanism is therefore triggered. The decreasing playback lag seen in s4 as playback position advances shows that blocks are often being skipped.

4.3.2 Skipped Blocks

Blocks may be skipped during playback according to each policy algorithm, in particular if there are blocks further ahead in the buffer that could be

played immediately. Figure 4.8 compares the mean share of skipped blocks under the playback policies and parameters investigated. Error bars indicate 95 percent confidence intervals of the means.

The share of skipped blocks is, as expected, generally inversely proportional to the playback lag shown by each policy, hence `sk-0` and `ca` policies skip more blocks than other policies. While, in Scenarios s1 and s2, the `sk-0` policy skipped block rate is higher than `ca`, the opposite happens in s3 and s4, since the catchup mechanism is triggered more often, due to the increased likelihood of content unavailability. With the use of the `rd` policy, though, the rate of skipped blocks is close to zero, but playback lag still varies, as seen in Section 4.3.1. This fact is due to the variable buffer size employed, that alters initial buffering. The remaining policies have their share of skipped blocks highly influenced by their respective parameters. On every scenario, under the `sk`, `rd`, `re`, and `ra` policies, higher values for the parameters used produce lower values of skipped blocks.

User-experienced image degradation levels vary according to specific video encoding and decoding algorithms (codecs) used – to which LiveShift is agnostic by design, as described in Section 3.2.1. Understanding both expected levels of skipped blocks and codec characteristics is thus crucial when choosing the appropriate playback policy for a specific situation. For instance, `sk-0` policies may only be employed in a situation in which the codec and user tolerate a skipped block rate roughly between 3 and 4%; the `ca` policy, in turn, may be impractical on scenarios with shortage of available upload capacity.

4.3.3 Failed Playback Sessions

The share of sessions in which the peer stalls for such a long time that playback is considered failed represent less than 0.5 percent of all sessions in Scenario s1, as shown in Figure 4.9. In contrast, in Scenario s4, the mean oscillates between 9.5 and 13.5 percent in all scenarios. The overlapping 95 percent confidence interval error bars indicate that the share of failed play-

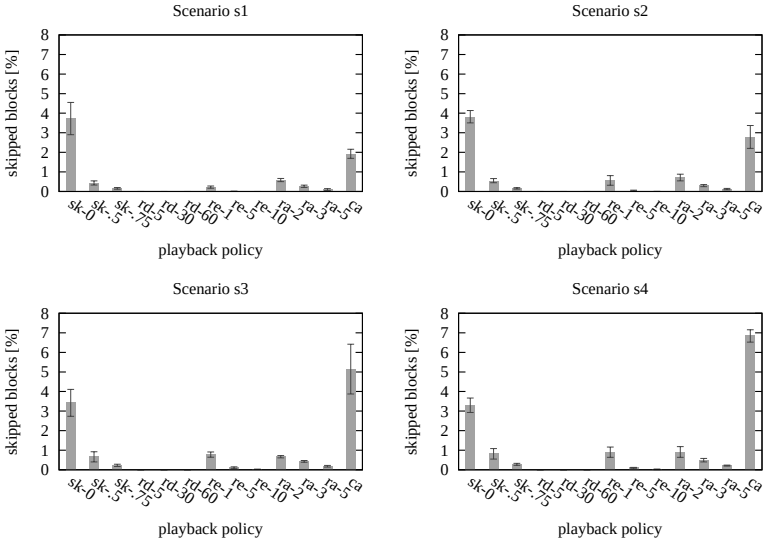

Figure 4.8: Skipped blocks

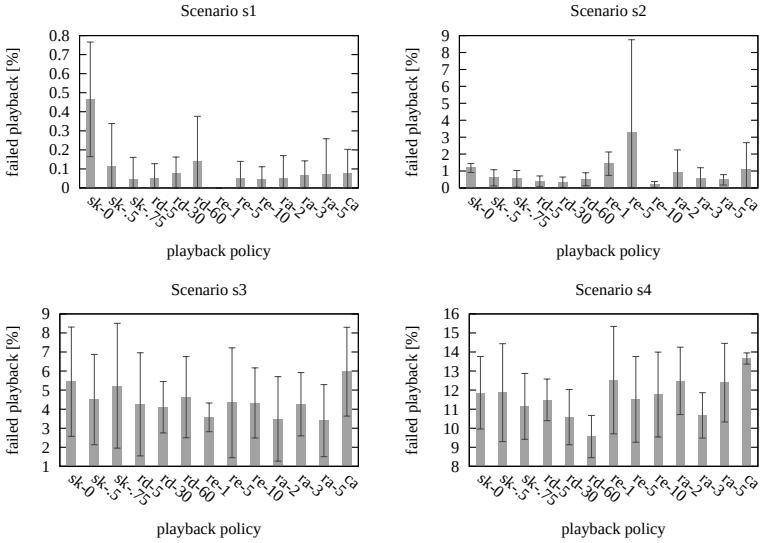

Figure 4.9: Failed playback

back depends rather on each scenario's available upload capacity than on the playback policy used.

4.4 Chapter Summary

This chapter has presented a detailed investigation on playback policies for P2P multimedia streaming systems. First, background and common terminology have been introduced to assist in defining the problem of playback policies. Then, a set of five playback policies has been formally defined with a uniform set of parameters for better comparability, using concepts both seen in related work and novel ones. These playback policies have been implemented in LiveShift, and realistic experiments have been executed in a testbed. Having observed the behavior of LiveShift under different playback policies, with different parameters, in scenarios ranging from under- to over-provisioned P2P networks, evaluation results made it evident that playback policies, though highly overlooked as part of the design of many P2P multimedia streaming systems, do affect user experience. Not only the selected policy, but also values used for its main parameters, affect important experience metrics, such as playback lag and share of skipped blocks.

5

B-Tracker: Improving Load Balancing and Efficiency in Distributed P2P Trackers

Trackers are important elements present in most P2P systems, providing the basic functionality of looking up peers that are providers for a specific resource. In LiveShift, that essentially means determining which peers offer any blocks in a certain segment – for a clarification about blocks and segments, please refer to Section 3.2.1. One of LiveShift's design objectives is benefiting from a fully-distributed architecture, as seen in Section 3.1, including scalability and robustness properties; hence, peers must share provisioning of the tracker service efficiently and uniformly, such that the load produced by the tracker service is evenly and fairly distributed.

However, as shown in Section 2.6, current implemented tracker approaches do not display efficiency and load balancing properties that would be appropriate in the LiveShift scenario, because some segments are expected to be significantly more popular than others. Thus, specific tracker entries for those popular segments will be looked up much more often than others. While there are several approaches for improving load balancing of

Table 5.1: B-Tracker nomenclature

b_p	Primary tracker provider limit per resource
r_p	Number of primary trackers a resource-provider mapping is written to
r_s	Number of secondary trackers a resource-provider mapping is written to
n_p	Primary trackers initially read for a resource

DHTs, they are generic in nature, thus a movement cost is always associated with content replication for load balancing.

Thus, this chapter introduces the design and evaluation of B-Tracker (Balanced Tracker) [50, 51], which is highly suited for the fully-distributed architecture of LiveShift. B-Tracker differs from existing work by (a) using a purely pull approach, which is more suitable than push for a tracker, since peers do not always require new providers; (b) replicating tracker entries proportionally to their popularity, inherently improving load balancing; (c) eliminating movement cost, since tracker data is moved only when peers request new providers, thus there is no cost for simply moving data; and (d) employing Bloom Filters to further improve efficiency by eliminating redundant entries from tracker messages. Results obtained are applicable to any P2P system that employs a tracker for provider discovery, including file sharing and multimedia streaming.

The remainder of this chapter is organized as follows. The B-Tracker approach is described in Section 5.1. Evaluation details and results are presented in Section 5.2. Finally, Section 5.3 displays final remarks.

5.1 B-Tracker Design

Though the terms *peer*, *tracker*, *provider*, and *neighbor* all refer to a participant in the P2P system, the use of such terminology defines more precisely the different roles that participants perform; every participant is expected to perform all roles in different situations. The terms are defined as follows:

a *peer* queries a *tracker* to obtain a list of *providers*, which are contacted directly for detailed availability information. If contact is successful and there is mutual interest, the peer and the provider become *neighbors* and are enabled to perform actual resource provisioning, that is, transfer data. Table 5.1 contains quantities used throughout this chapter.

The basic functions a tracker offers to peers are the following:

- *getProviders(resourceID)*, which returns a (predefined-sized) set of providers for the resource,
- *addAsProvider(resourceID)*, that adds the sender of the message to the provider set of the given resource, and
- *removeAsProvider(resourceID)*, called by a peer that is not anymore a provider for the given resource.

It is assumed that, once a peer is provided with a resource, e.g., a file or a video stream, it becomes itself a provider for it.

5.1.1 Primary Trackers

At first, B-Tracker employs a distributed hash table (DHT)-like structure for initial tracker discovery, since DHTs offer a scalable structure to store key-value mappings at well-known locations. Peers with *peerID* closest to the key (the *resourceID*) are responsible for storing the list of providers for the resource. These peers – termed *primary trackers* – are discovered in $O(\log n)$ messages, where n is the number of peers in the system [84]. In order to enable a DHT to be used as a tracker, its original *put(key, value)* function is modified to allow multiple tuples (*peerID, IP address, TCP or UDP port* of providers) to be stored under a single key, and *get(key)* is adapted to return a random subset of these tuples.

The number of peers that are primary trackers for a resource is given by the primary tracker replication factor r_p. Since primary trackers are probably not providers for the resources they track, due to the random distribution of keys based on uniform hashing in a DHT, and to motivate peers to use secondary trackers, a limit of b_p providers per resource is set on the provider storage capacity.

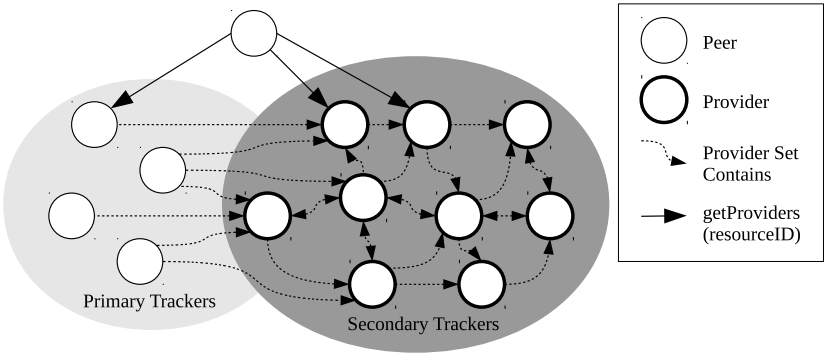

Figure 5.1: Primary and Secondary Trackers

5.1.2 Secondary Trackers

Once a peer has obtained a provider list from a primary tracker, subsequent tracker queries can be issued to any provider, since each of them is a *secondary tracker* for the resources it provides. The concept of secondary trackers is introduced to improve both scalability, since resources with more providers are able to distribute the load among more trackers, and fairness, because the load is shared by those peers interested in providing the resource. In other words, the number of secondary trackers scales with demand, offloading the primary trackers.

Figure 5.1 illustrates how B-Tracker uses primary and secondary trackers to discover providers. Primary trackers may be, coincidentally, as well providers for a particular resource. Each primary tracker provides bootstrapping to the graph formed by secondary trackers, which are all providers for a particular resource. Any primary or secondary tracker sends their provider set as a reply to the *getProviders(resourceID)* message.

The parameter n_p sets the number of primary trackers that peers consult before requesting from a secondary tracker, when *getProviders(resourceID)* is called. While a low value increases load balancing due to the load decrease of primary trackers, it increases the likelihood that returned peers have none or outdated information about additional providers. This happens because secondary trackers are less reliable, since they cease being

trackers for resources as soon as they are not anymore part of the respective swarms.

Finally, secondary trackers are not limited in storage capacity, since tracker data is relatively small and write operations are expected to be well distributed, due to the random distribution of secondary tracker discovery.

5.1.3 Improving Efficiency

The Coupon Collector's Problem [90] states that, if at each step a random sample from a set of size n is drawn, the number of steps necessary to retrieve all set members grows as $O(n \log n)$. This problem hinders the functionality of current P2P trackers, since they are based exactly on drawing a random sample from a set of providers.

In order not to suffer from the Coupon Collector's Problem, while avoiding that each tracker keeps state about all providers supplied to each request, a solution based on Bloom filters [11] is used. Queries to primary and secondary trackers include a Bloom filter with all already known providers. Trackers take the filter into consideration by returning a random subset of known providers for the specified resource, excluding providers that match the filter.

For example, if a peer p is searching for new providers, and it already knows providers p_1 and p_2, it issues tracker queries to tracker t containing a Bloom filter with $\{p_1, p_2\}$. Tracker t excludes $\{p_1, p_2\}$ from its reply, replying only with information about how to reach other providers, for instance p_3 and p_4.

While Bloom filters save bandwidth due to their fixed size, they may produce false positives [11]. This means that an unknown provider may not be found. The probability of false positives can, though, be adjusted to be low, by considering the filter length and expected number of items, as described in Section 5.2.1.

5.1.4 Updating Mechanism

The *addAsProvider(resourceID)* operation is used by peers that have become providers for a certain resource to announce that fact to other peers. This

causes the distributed resource to provider map to be updated. The operation can be issued to both primary and secondary trackers, though primary trackers accept only up to b_p providers per resource, as discussed in Section 5.1.2.

While primary trackers are a fixed set of peers found via the DHT logic, secondary trackers are much more numerous for larger swarms. Different strategies could be used by peers for choosing the subset of secondary trackers to send *addAsProvider(resourceID)* messages. B-Tracker implements a *stable random* approach, which consists on initially selecting a random subset of known secondary trackers to announce new resources, and always using the same subset, if possible.

The reason why this random subset is kept stable for a particular resource is to reduce state kept at peers and the number of messages that are required to be sent during a *removeAsProvider(resourceID)* operation, since providers keep track of which trackers they have announced themselves at. As a result of churn, though, some trackers may leave the system or become unreachable, in which case they are replaced by new random trackers.

The replication factor r_p determines on how many primary trackers information is stored, while r_s refers to the number of replicas at secondary trackers. A higher r_s value increases the chance that a provider is found, while update and maintenance operations are more expensive.

5.1.5 Outdated Information

Tracker information tends to become outdated as peers fail, leave, or stop providing resources, without informing the responsible trackers. This is a problem not only for distributed tracker architectures, but as well for centralized ones. Having the tracker verify all providers it holds is too large an effort due to the potentially large number of entries. Hence, B-Tracker assigns a time to live (TTL) to each resource-provider mapping stored. Providers need to update (via *addAsProvider(resourceID)*) their respective tracker entries before the TTL runs out.

A lower TTL value increases the number of *addAsProvider(resourceID)* messages in the system, but reduces the number of outdated tracker en-

tries. TTL is currently a globally-known value that is set to 60 minutes. An optimal TTL value depends on churn levels, as well as the overall tolerance to outdated information, and could be adapted on-the-fly [8]. Such optimization is, however, left for future work.

5.2 Evaluation

B-Tracker is a novel tracker approach that can be used by any P2P system that uses a tracker, for instance, multimedia streaming (e.g. LiveShift) and file sharing (e.g. BitTorrent) systems. Evaluations are, therefore, performed in a neutral environment that isolates tracker queries.

Evaluations are done with a Java 1.6 implementation of B-Tracker to analyze its properties in comparison to other distributed tracker approaches, namely PEX and pure DHT-based trackers. All investigated trackers were implemented as part of the TomP2P [115] project, namely the DHT, PEX, and B-Tracker approaches. Section 6.3 describes the implementation of B-Tracker in more detail.

The evaluation focuses on efficiency and load balancing, two important metrics for any P2P system. Load is defined in terms of upstream traffic, since it is the scarcest resource in a file-sharing or multimedia streaming P2P system. *Efficiency* is defined in terms of mean load per peer in the swarm, considering all tracker-related messages sent, so less load conveys better efficiency. *Load balancing* is defined as the standard deviation of load among all peers in the swarm, so less deviation determines a better balance.

5.2.1 Evaluation Parameters

The parameters used for the evaluation are as follows. The Bloom filter assumes a probability of false positives $p = 0.0073$ with a number of items $n = 100$, which result in a filter of size $m = 1024$ bits [11] – a good compromise under realistic assumptions. A fixed replication factor $r_p = 20$ is used for the DHT approach, as in the popular BitTorrent implementation [26]. B-Tracker uses $r_p = 2$ and $r_s = 18$ for replication, since they add up to 20, in order to be comparable in a fair manner to the DHT approach. The number

of primary trackers that peers consult before requesting from a secondary tracker $n_p = 0$, that is, they always query secondary trackers for providers first, resorting to primary trackers only if all queries to secondary trackers fail – this is to maximize load balancing. Primary tracker storage capacity $b_p = 35$ providers per resource, since 35 is a common number of neighbors used by P2P applications. All results were obtained from 100 runs.

A P2P system with 1000 peers was emulated as follows. At each run, a swarm is initially created with 50, 250, or 450 peers. Peers in the swarm are interested in obtaining a certain resource, e.g., downloading a file. Each peer in the swarm obtains 35 providers from the DHT. Measurement starts only after they have obtained those initial providers, in order to emulate a live swarm. The system, then, suffers from churn, which is defined as the percentage (10, 20, 30, or 40 percent) of peers in the swarm that go offline, being immediately replaced by the same number of newly created peers. All peers attempt in turn to have again 35 providers in total, exchanging messages according to the tracker approach being analyzed.

In the DHT approach, peers query always a random one of the 20 peers that are responsible for holding the provider list for the resource in question. The tracker always replies with a random subset of at most 35 providers.

In the PEX approach, peers exchange PEX messages containing a set with newly added neighbors and a set of disconnected neighbors. If, after exchanging PEX messages, a peer still does not have 35 providers, it queries the DHT to obtain them.

In the B-Tracker-NF – NF stands for *no Bloom filters* – approach, peers query one or more random secondary trackers, which in essence are providers obtained from the initial DHT call until they obtain at least 35 providers. If, after querying all known secondary trackers, a peer has not reached its goal, it queries a primary tracker to obtain them.

The B-Tracker approach works like B-Tracker-NF, except that all requests contain a Bloom filter holding the currently known providers.

5.2.2 Efficiency

In a more efficient system, the knowledge of which are current providers is spread with less traffic generated per peer. Efficiency is, therefore, defined in terms of the average load per peer; load being defined as bytes sent per peer, on average.

Figure 5.2 shows the average load per peer for swarm sizes 50, 250, and 450, respectively. On all bar plots of this chapter, each value shown is an average of all runs, with error bars displaying the standard deviation. As seen in all plots, in general, B-Tracker achieves better efficiency when compared to pure DHT and PEX approaches. A DHT approach is not efficient because each new peer and every peer with less than 35 providers in the swarm need to query the DHT, which creates many routing messages. PEX is even less efficient, because it requires that peers send many unnecessary messages, informing neighbors about their new neighbors regardless of whether or not it is needed. B-Tracker shows better efficiency than B-Tracker-NF due to the use of Bloom filters – though request messages are larger, since they contain the filter, the provider list returned by trackers contains only useful information, further improving overall efficiency.

Concerning churn, Figure 5.3 shows that load increases with churn for all investigated approaches, since, with more churn, there are more newly created peers that look for providers, and more peers need to obtain more providers. PEX, however, produces a higher load increase with higher churn when compared to the other approaches, because PEX messages contain the list of neighbors that were added or removed, which grows larger with churn.

Figure 5.3 displays efficiency per churn level to ease comparison of the different approaches as the swarm size increases. The B-Tracker approach shows better scalability, since, for larger swarms, the mean peer load increases only slightly. DHT and PEX experience a larger load increase from swarm size 50 to 250, though from 250 to 450 it increases only slightly. The difference between B-Tracker-NF and B-Tracker shows that using Bloom filters as proposed improves efficiency, especially with larger swarm sizes.

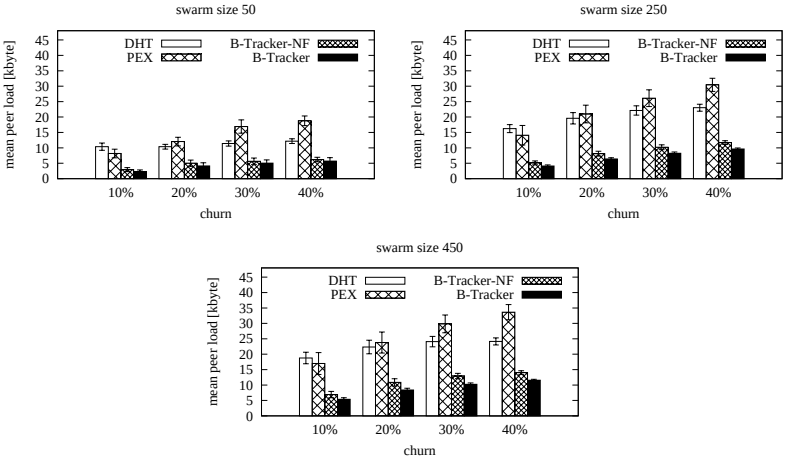

Figure 5.2: Efficiency per swarm size

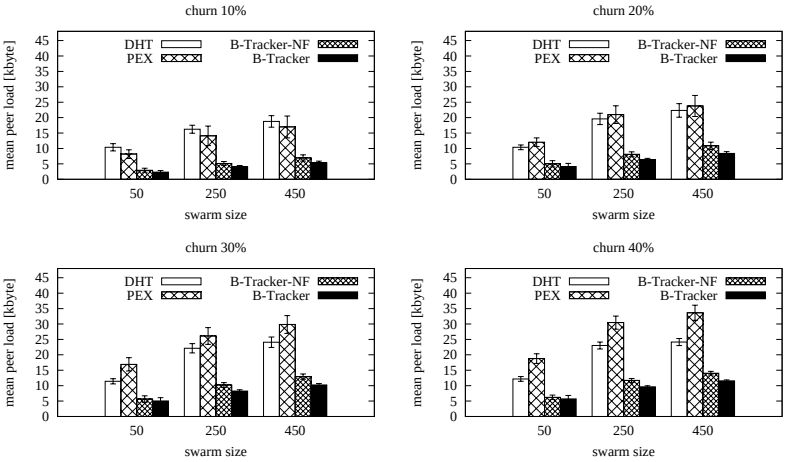

Figure 5.3: Efficiency per churn level

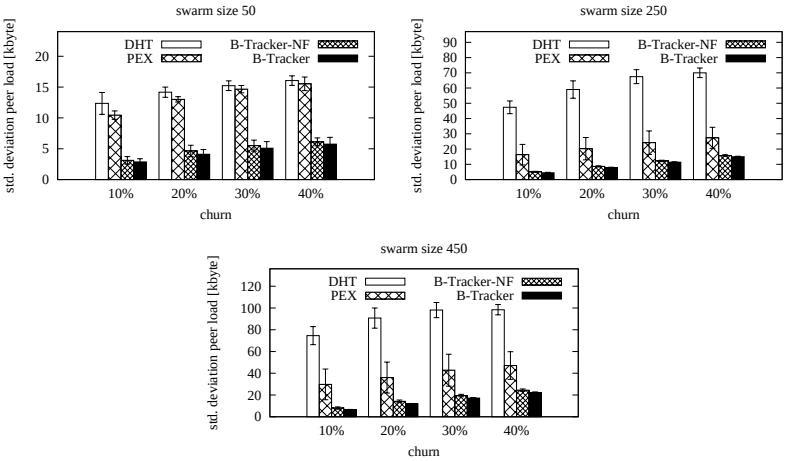

Figure 5.4: Load balancing per swarm size

For instance, with 40% churn and swarm size 450, when compared to a pure DHT approach, PEX has shown to increase tracker traffic by 39.28%. B-Tracker-NF has, though, reduced it by 41.83%. B-Tracker with Bloom filters has reduced tracker traffic by 52.10%, also relatively to pure DHT. The largest relative improvement in efficiency was 77.53%, observed with B-Tracker with swarm size 50 and 10% churn. The smallest relative improvement in efficiency with B-Tracker was still 52.10%, with swarm size 450 and 40% churn.

5.2.3 Load Balancing

Figure 5.4 displays load balancing as the standard deviation of load among all peers in the swarm. Load balancing was calculated for each run and an average for all runs is displayed; error bars show, then, the standard deviation for the different runs.

In all investigated scenarios, B-Tracker-NF and B-Tracker distribute load much better than DHT, due to the presence of secondary trackers. In a pure DHT approach, peers that are trackers become heavily loaded as swarm size

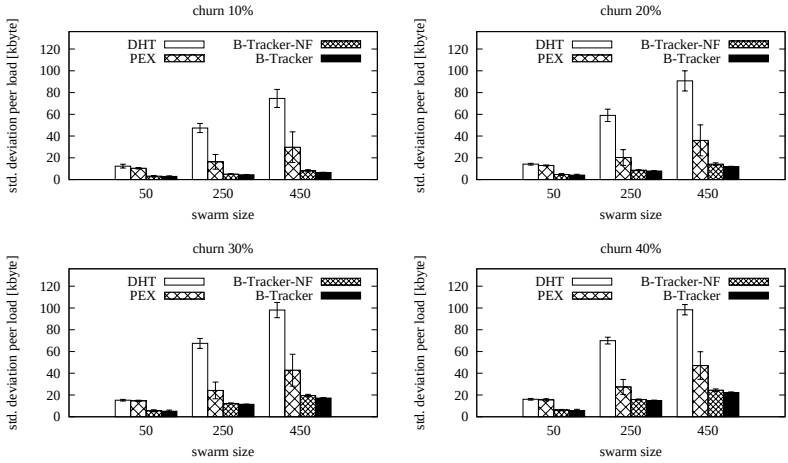

Figure 5.5: Load balancing per churn level

increases, i.e. with swarm size 450. PEX shows improved load balancing, especially with larger swarms. B-Tracker-NF shows that using Bloom filters as proposed improves load balancing only by a small amount. The fact that load balancing degrades with larger swarms on all approaches is explained by their use of DHT for initial tracker discovery, besides as a last resort if PEX and secondary trackers do not yield the goal of 35 providers per peer.

Churn has a negative influence on load balancing in all investigated approaches and swarm sizes, as Figure 5.5 pictures. This is also due to the DHT being queried at least initially by all new peers. The difference between load balancing values, however, is small between 30 and 40 percent churn rates, showing that it increases at smaller steps.

For instance, with swarm size 450, when subject to 40% churn, PEX improves load balancing of pure DHT already by 52.18%. B-Tracker-NF shows even better load balancing, resulting in 75.16% improvement in load balancing, also compared to DHT. B-Tracker with Bloom filters in this case increases the index to 77.30%. The largest relative improvement in load balancing was 91.30%, observed with B-Tracker with swarm size 450 and 10% churn. The smallest relative improvement in load balancing with B-Tracker was still 64.16%, with swarm size 50 and 40% churn.

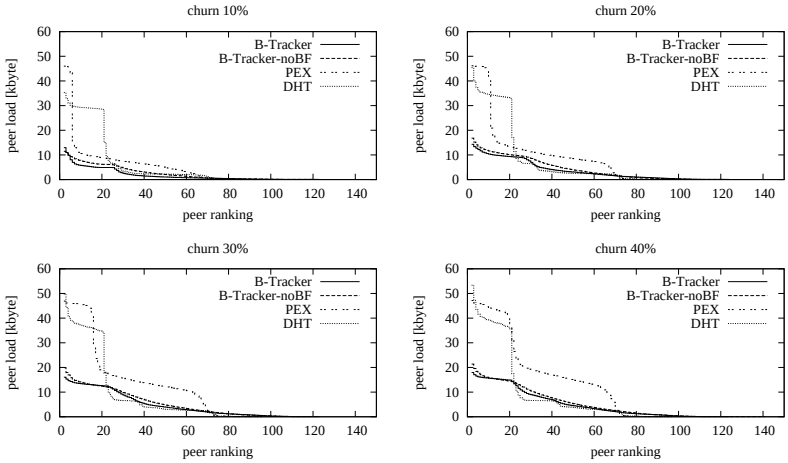

Figure 5.6: Peer load, swarm size 50

5.2.4 Detailed View

Figures 5.6-5.8 show in more detail the peer load in the different scenarios studied and improve the understanding of results. An average of the top 150 peers with highest load on the different runs is shown. Load is caused by sending messages, including both request and reply messages, and DHT routing and keep-alive messages.

It can be seen that the top 20 peers have a significantly higher load than the others, due to them being responsible for the holding the DHT values of the resource sought after. The peak in peer load observed for the first 1-5 correspond to the peers that, besides being responsible for tracking providers, also issue messages to find providers.

In PEX, load is much better distributed than on the pure DHT approach, but there is still a peak on the most loaded peers. This happens because PEX messages are often not enough for peers to again discover 35 providers, so they still need to resort to the DHT frequently. This happens because PEX alone does not always spread provider information to all peers, which end

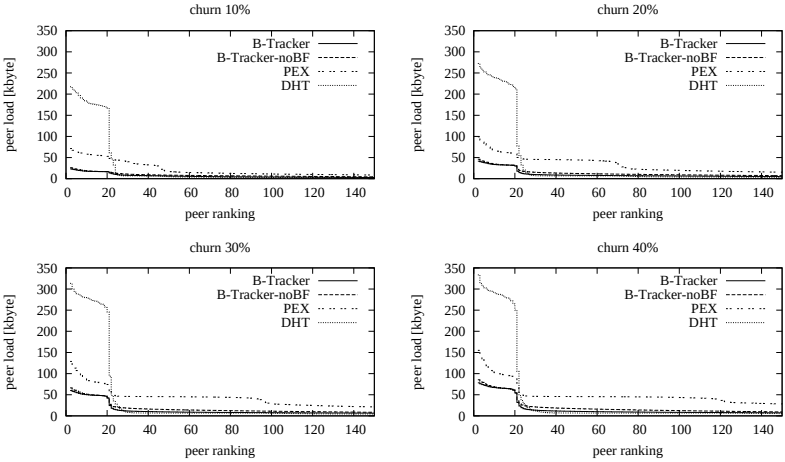

Figure 5.7: Peer load, swarm size 250

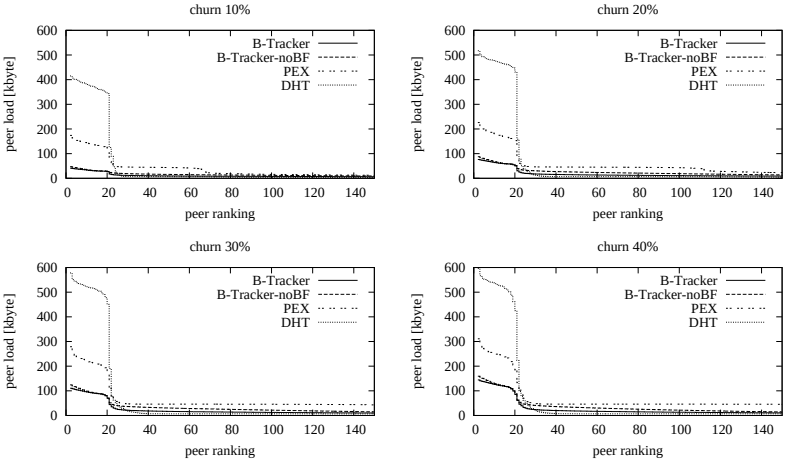

Figure 5.8: Peer load, swarm size 450

up having to resort to the DHT. In general, PEX shows better load balancing with low (10% and 20%) churn level. B-Tracker-noBF further spreads load to more peers, largely reducing the load of the top 20 loaded peers. Higher churn (30% and 40%), though, still increases load unevenly among peers.

It can also be seen that B-Tracker with Bloom filters decreases peer load approximately evenly compared to B-Tracker-noBF. This confirms what was observed in Sections 5.2.2 and 5.2.3, that the use of Bloom filters as suggested does improve efficiency, but load balancing only slightly.

5.3 Chapter Summary

This chapter has presented and discussed the design and evaluation of B-Tracker, a novel P2P tracker that improves the state of the art in the area. The main innovation of B-Tracker is that all peers are trackers for the resources they provide, thus better distributing the load of popular resources. Extensive evaluations have confirmed B-Tracker's ability, in comparison to other distributed P2P trackers, namely a pure DHT and PEX approaches, to improve both efficiency and load balancing. The B-Tracker approach is particularly suited for large swarms and high churn rates. Using Bloom filters as designed brings an extra improvement in efficiency.

6

LiveShift Application

The LiveShift system has been fully implemented as a Java [63] application, in order to (a) validate its concepts in a practical environment, (b) allow demonstrations, such as [12, 48], (c) facilitate comparative testing of different policies by the research community, and (d) enable the release of the application as open source software. Besides, with a full-fledged implementation, it is possible to obtain highly realistic results in comparison with sole simulations, since there are less input values that are given, and more that are generated by the system itself. The application's source code has been published under the GNU General Public License (GPL) version 3.0 [2] on GitHub [78], so the both the research community and end-users may benefit from it. Hence, this chapter presents an insight into LiveShift's application. First, the graphical user interface (GUI) is presented, with screenshots and clarification on how it is operated by a user, relating its functionality to its architecture components. Then, the implementation and integration of playback policies are presented, arguing for the modular and flexible implementation of policies in LiveShift. Finally, B-Tracker's implementation and integration to LiveShift are highlighted.

Figure 6.1: LiveShift's main screen, when connected

6.1 LiveShift's Graphical User Interface

The initial frame of LiveShift's graphical user interface (GUI), as designed and implemented, is shown in Figure 6.1. It corresponds to the GUI component seen in Figure 3.1. The interface is designed to be simple, intuitive, yet informative. On the left-hand side of the screen, the main buttons allow control of LiveShift's top-level functionality. From top to bottom, the user interface offers users the following:

- connect (i.e., bootstrap) to the P2P network, by using a well-known peer or a DNS mapping, such as `boostrap.liveshift.net`;
- publish a new channel, e.g., from a device, such as a camera, or a pre-recorded media file;
- toggle visibility of the channel list on the right-hand-side of the frame;
- adjust configuration settings, including network, video encoders and players, and storage; and

- viewing statistics and details about the P2P network, including blocks held, skipped, and scheduled for download, messages exchanged with other peers, granted upload slots, and peers in the upload slot queue.

Any user is able to publish a new channel. Figure 6.2 displays the Publish frame, with important configuration of the new channel. Name and description are used to describe the channel content, and its words are split to be used for searching. Currently, the following three sources are supported by the VLCJ [119] implementation in use:

- a file, e.g., on the user hard drive;
- a device, e.g., a Digital Video Broadcast (DVB) tuner, webcam;
- or network stream, e.g., a multicast stream being received, an RTP or RTSP stream.

Specific protocols and transcoding options may be optionally provided, and depend on VLCJ support. When the Publish button is pressed, the Encoder and Assembler components (*c.f.* Figure 3.1) are launched in the background to produce and announce blocks and segments with the new streams.

The Settings frame, displayed in Figure 6.3, allows the user to adjust application configurations. Among them are all network configurations, including network interface, port number, peer name, upload rate, time server, and bootstrap peer address.

More advanced settings are set exclusively via command-line parameters. These include, for example, selecting playback policies and their parameters, setting initial buffering and defining buffer sizes, as well as choosing and tuning other policies.

As soon as a connection to the P2P network is established, the right-hand side of the frame displays a list of available channels. Since the list is expected to grow to a large number of channels, a search box is included. The search box queries the P2P systems for channels that match

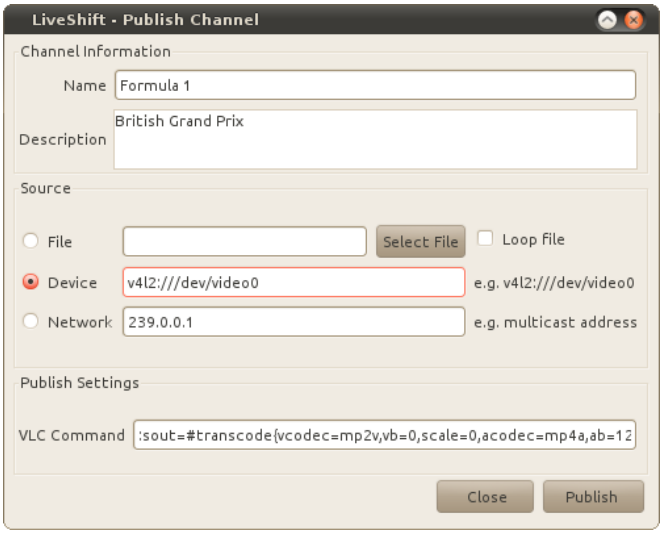

Figure 6.2: LiveShift Publish frame

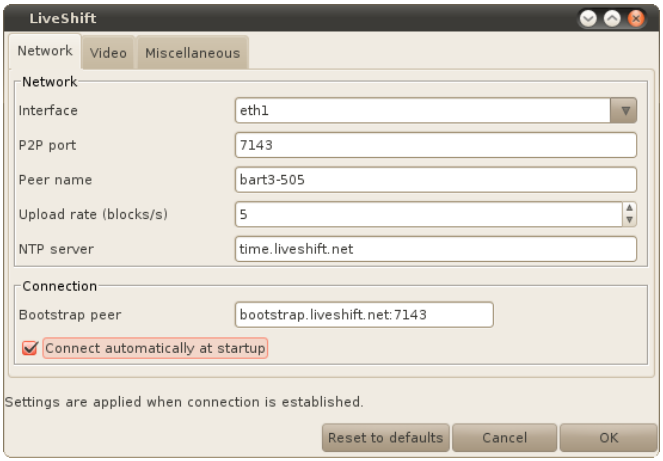

Figure 6.3: LiveShift Settings frame

Figure 6.4: LiveShift main screen, channel selected

the searched keywords. The channel creator defines a number of keywords, or *tags*, that can be used for searching later by the viewers.

The larger central area of the frame is where the video is played back to the user. As soon as a channel is selected, the playback control bar is revealed, as shown in Figure 6.4. Buttons, from left to right, allow users to control playback as follows:

- rewind by skipping 5 seconds backward (holding the button increases the amount of skipped blocks), thus increasing playback lag;
- pause playback by freezing image but continuing downloading and buffering blocks in the background (limited by the download block selection policy, *c.f.* Section 3.4.2), also increasing playback lag;
- stop playback by stopping downloading this stream entirely;
- fast-forward by skipping 5 seconds forward (holding the button also increases the amount of skipped blocks), thus decreasing playback lag as possible;

Figure 6.5: LiveShift main screen during playback

- start playback from the current date and time, thus viewing the live stream;
- select an arbitrary date and time to start playback, thus viewing a time-shifted stream;
- display the video in full-screen mode;
- increase volume; and
- decrease volume.

Tooltips are in place to further help the user interpret those buttons correctly.

Figure 6.5 shows LiveShift in full operation, after a channel has been selected and the play button has been pressed. The center part of the window is occupied by the Player component (*c.f.* Figure 3.1), provided by the VLCJ [119] library, to decode video blocks that are supplied by the Player Sender component. The status bar at bottom part of the frame displays information about network connectivity, publishing status, and playback details, including the channel name, whether it is playing, paused, or stopped

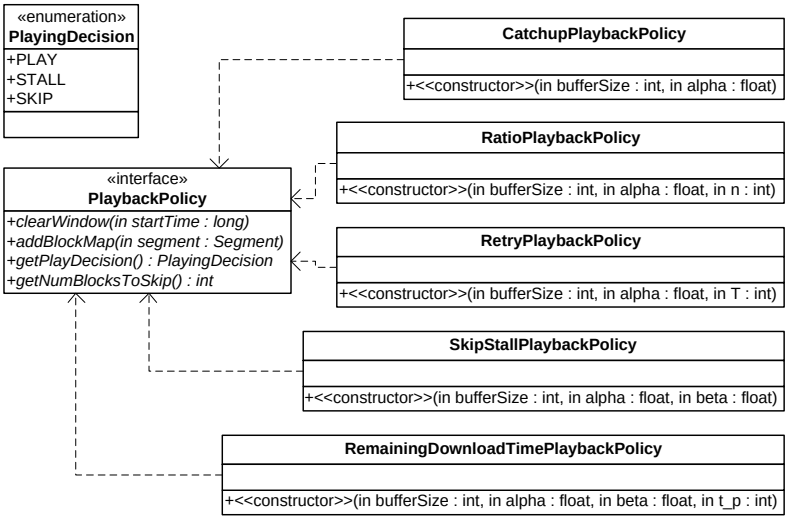

Figure 6.6: LiveShift main screen during playback

(in square brackets), whether the playback policy has determined the current block to play, stall, or skip, and date and time being played back. Status information is obtained from the Tuner component.

6.2 Implementation of Playback Policies

All playback policies defined in Section 4.2 have been implemented into LiveShift. According to LiveShift's architecture as pictured in Figure 3.1, playback policies are implemented in the Player Sender component, which is responsible for retrieving blocks from the Storage and sending them to the Player to be decoded and displayed to the user.

Figure 6.6 shows a UML class diagram, illustrating how playback policies are implemented in LiveShift. The interface `PlaybackPolicy` defines all important methods to be implemented by the Player Sender, namely:

- `clearWindow()` initializes the object with the current playback position.

- `addBlockMap()` supplies a segment, so the policy can look which blocks are held in storage, and which are not. May be called twice if playing near the end of a segment, with the next segment as parameter.

- `getPlayDecision()` based on the block maps added, this method performs the playback policy decision, returning either PLAY, STALL, or SKIP.

- `getNumBlocksToSkip()` in case SKIP was the playback decision, this method returns the number of blocks to be skipped.

- constructors: each implementation defines its corresponding constructors, according to their definitions.

This exemplifies the modular approach taken by LiveShift, which eases the implementation of novel policies. In order to implement new policies, only a class implementing those methods needs to be defined, and the corresponding command-line arguments passed when running the application.

6.3 Implementation of B-Tracker

According to Figure 3.1, the distributed tracker (DT) is a component in the LiveShift architecture. The distributed tacker functionality is provided by TomP2P [115] library, in which all investigated trackers (DHT, PEX, and B-Tracker) were implemented.

The interface between the DT and the rest of LiveShift is relatively simple and straight-forward; only the following two methods are defined:

- `publishSegment(segmentIdentifier)` announces on the DT that the peer can provide at least one block in the respective segment.

- `getPeerList(segmentIdentifier, howMany, exclusionFilter)` looks up the DT and returns, for the respective segment , the number of providers specified by the `howMany` parameter.

6.4 Chapter Summary

This chapter has presented implementation details of the LiveShift application, a flexible and modular application that allows the investigation of P2P mechanisms for both live and on-demand multimedia streaming. Designing and implementing such application, while being technically challenging, allows for a realistic evaluation of the involved algorithms, while enabling obtention of results that complement pure theoretical simulation ones. Further, this chapter has illustrated the LiveShift GUI as a simple, objective, and informative interface used to operate its main functions. This chapter has, as well, displayed implementation and integration details of playback policies and B-Tracker in LiveShift, with UML diagrams and a description of its main methods. All source code has been made available on GitHub [78], which can be used for further research activities.

7

Conclusions

This thesis has proposed and investigated several key aspects involved in allowing a fully-distributed and seamless integration of live and time-shifted P2P multimedia streaming, a use case that was largely unexplored before this thesis. It represents an important step in reaching this goal, by focusing on four important aspects, namely (a) developing and evaluating a fully-distributed, mesh-pull P2P *protocol* that allows the exchange of both live and time-shifted video streams, and defining a set of decoupled *policies* that influence the behavior of the protocol; (b) developing and evaluating a reference *client implementation* that allows a complete and realistic investigation of proposed advancements; (c) investigating *playback policies* under different scenarios and parameters, including live and on-demand characteristics; and (d) defining and evaluating a *fully-distributed P2P tracker* that improves state-of-the-art regarding both efficiency and load balancing. This thesis has shown the above mechanisms' practicability by integrating them in a single, real application, called LiveShift.

7.1 LiveShift Architecture, Protocol, and Policies

In Chapter 3 a flexible and fully-decentralized mesh-pull P2P protocol for locating and distributing both live and time-shifted video streams in an integrated manner has been proposed, as well as policies to be used with the new protocol, revealing and discussing the main trade-offs encountered in building such a system, as well as evaluation results.

Trace-driven evaluations focus in scenarios with many channel switches and an increasing number of peers with upstream capacity lower than the rate of the video stream being transmitted. The system shows its ability to find content and upstream capacity quickly enough to sustain a low playback lag relatively to their playback position. LiveShift also shows ability to combine the upload capacity of several low upstream peers on time. In the scenarios studied, the system supports a low (less than 10 s) playback lag for 80 percent of users with high bandwidth, even in the presence of churn – in form of channel switching, time shifting, and peers disconnecting. Users with low upstream bandwidth are negatively affected in scenarios with high churn or overall low upload capacity, but playback lag remains within 60 seconds of transmission for over 95 percent of the peers in the investigated scenarios. Finally, in the scenarios studied, overhead remains below 3.1 percent of the original bitrate of the video stream being transmitted.

This shows that the mesh-pull protocol introduced by this thesis, including division of the stream in time-addressed blocks and segments, backed by distributed hash table and distributed tracker structures, running the proposed policies, is capable of successfully supporting the desired use case in the investigated scenarios. The application has been published [78] with an open-source license [2] as a contribution to the scientific community.

7.2 Playback Policies

The behavior of LiveShift under different playback policies, with different parameters, in scenarios ranging from under- to over-provisioned P2P networks, has been shown in Chapter 4. It is evident that *different playback*

policies do *affect user experience in a P2P video streaming system*, in terms of both playback lag and share of skipped blocks.

This raises the second research question in the scope of playback policies, *which playback policies are most suitable for live and on-demand scenarios?* Under circumstances in which minimizing playback lag is the main goal, which might be desirable by viewers of live (e.g., sports) events, the Catchup (`ca`) and the Always Skip (`sk-0`) policies are the most suitable policies studied, considering that they have consistently shown much lower playback lag for a majority of peers compared to all other approaches. This comes, however, at the cost of a higher number of skipped blocks.

If lowest number of skipped blocks is the objective, the policies that have shown to skip less than 0.5 percent of the total blocks on both scenarios s1 and s4 are the Skip/Stall policy with $\beta = .75$ (`sk-.75`), the Retry policy with $T \geq 5$ (`re-5` and `re-10`), the Ratio policy with $n \geq 3$ (`ra-3` and `ra-5`), and the Remaining Download Time policy (`rd`). These policies may be applied in cases in which occasional interruptions are of less importance than skipping content, for instance for VoD.

Alternatively, compromising playback lag and skipped block rate may be the goal. Policies that show a skipped block rate inferior to 0.5 percent and playback lag inferior or equal to 45 seconds at playback position 15 minutes (for 80 percent of peers) are, on the under-provisioned Scenario s4, the following: the Skip/Stall policy with $\beta = .75$ (`sk-.75`), the Retry policy with $T \geq 5$ (`re-5` and `re-10`), and the Ratio policy with $n \geq 3$ (`ra-3` and `ra-5`). In Scenario s1, the Skip/Stall policy with $\beta \geq .5$ (`sk-.5`, and `sk-.75`), the Retry policy with $T = 1$ (`re-1`), and the Ratio policy with $n = 3$ (`ra-3`) are policies that yield a skipped block rate inferior to 0.5 percent and playback lag less than or equal to 9 seconds at playback position 10 minutes, also for 80 percent of peers.

Understanding the behavior of playback policies is, hence, imperative to select the most appropriate policy for the desired result, whether it is keeping playback lag as low as possible, avoiding skipping many video blocks, or achieving a compromise. The ultimate decision may be left completely up to the user, or depend on the type of content being transmitted.

7.3 B-Tracker

B-Tracker, a pull-based, fully-distributed P2P tracker, has been introduced in Chapter 5. B-Tracker improves load balancing by increasing the number of replicas proportionally to content popularity. Its pull approach eliminates providers being sent to peers which are not interested in receiving new providers. The pull approach also allows the use of Bloom filters to eliminate irrelevant providers from tracker replies, which is not possible in push approaches, such as PEX.

Extensive evaluations show that B-Tracker achieves better load balancing and higher efficiency than other distributed trackers in all investigated scenarios. B-Tracker improves efficiency from 52.10% to 77.53%, and load balancing from 64.16% to 91.30%, relatively to a pure DHT approach. A pure DHT approach shows poor load balancing because it uses a fixed replication factor, so peers responsible for storing tracker information of popular keywords and files tend to get overloaded. PEX shows improved load balancing, since it offloads the DHT, but lower efficiency, because peers exchange messages which may not be of interest. Finally, a larger swarm size and higher churn produce only small degradation in B-Tracker's both load balancing and efficiency. The use of Bloom filters as suggested helps a further increase in system efficiency by avoiding redundant traffic, particularly in large swarms that suffer from high churn.

B-Tracker shows that relying on direct exchange of data that already are at peers is a simple yet successful approach to increase fairness in P2P systems. Also, a pull approach is more efficient in this case, since peers do not always need new providers, and when they do, they shall be able to issue a request immediately.

7.4 Evaluation of Design Objectives

The investigation and development of the novel approaches in this thesis were largely guided by the design objectives defined in Section 3.1, which were reached as follows.

1. **Free Peercasting:** By design, any peer is able to publish a channel;

2. **Scalability:** A growing number of peers results in playback lag below 60 seconds of transmission for over 95 percent of the peers in the investigated scenarios;

3. **Robustness:** Churn in form of peers joining and leaving the system and switching both channels and positions in the time scale is tolerated, and its negative effects can be managed by employing the appropriate playback policy;

4. **Full decentralization:** DHT and DT are employed as scalable, robust, and efficient distributed data structures; and

5. **Low overhead:** The introduced network overhead remains below 3.1% in the scenarios considered.

7.5 Future Work

While the definition and evaluation of protocol, policies, and tracker represent an important first step into supporting the proposed use case of integrating both live and time-shifted video streaming in a fully-decentralized environment, open research questions still remain.

Future work involving LiveShift protocol and policies includes further analyzing and finding optimal policies, for example the storage policy, which has not been extensively explored at present. Additionally, the introduction of ALTO [70, 93] capability for locality-awareness of both download and upload peer selection is a highly-promising research direction. Also, an effective incentive mechanism must be investigated to verify the upload capacity of peers that may be applied in the proposed use case, which involves asymmetry of interest in a time-constrained environment. Orthogonally, investigation on existing and novel transport protocols, e.g. LEDBAT [120], are promising to reduce playback lag in event of congestion. Finally, running a global-scale trial with a higher number of users and channels would further validate LiveShift architecture, protocol, and policies, plus allowing subjective QoE values to be obtained. Further important

issues to be considered concern overall system security, i.e. preventing malicious users from harming other users.

Concerning the investigated playback policies, in the evaluated scenarios, all peers adopt a uniform playback policy, which allows evaluation of their effect on the entire distribution overlay, if assumed that all users are interested in either live or on-demand characteristics. Future work may investigate scenarios in which peers adopt mixed policies, which are likely in LiveShift. There is also the opportunity of combining characteristics of different policies. A further promising possibility is creating a predictive playback policy that considers past peer experiences to avoid stalling when the probability that a missing block is downloaded in a timely fashion is low. Finally, addressing novel QoE aspects [55, 56] would be a natural continuation of this work.

Regarding B-Tracker, future work will investigate security aspects involving malicious peers and trackers, that may have incentive to give out carefully-crafted, invalid replies to other peers. Besides, theoretical bounds for B-Tracker operations shall be established, to prove its scalability in very large systems. Another possible research direction involves improving locality-awareness of secondary trackers, to reduce average latency in obtaining providers. A final step would be deploying and evaluating B-Tracker on different P2P systems.

References

[1] *Dummynet - Network Emulation Tool for Testing Networking Protocols*, http://info.iet.unipi.it/~luigi/dummynet/, last visited: Februray 2012.

[2] *GNU General Public License, version 3*, http://www.gnu.org/licenses/gpl.html, last visited: October 2012.

[3] *ITU-T Recommendation G.1030. Estimating End-To-End Performance in IP Networks for Data Applications*, November 2005.

[4] Eytan Adar and Bernardo A. Huberman, *Free Riding on Gnutella*, First Monday, Internet Journal **5** (2000), no. 10.

[5] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T. Suel, and D.D. Yao, *Optimal Peer Selection for P2P Downloading and Streaming*, 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005) (Miami, FL, USA), vol. 3, Mar. 2005, pp. 1538–1549.

[6] R. Axelrod and W.D. Hamilton, *The Evolution of Cooperation*, Science **211** (1981), no. 4489, 1390–1396.

[7] R. Bindal, Pei Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, *Improving Traffic Locality in BitTorrent via Biased Neighbor Selection*, 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006) (Lisbon, Portugal), Jul. 2006, pp. 66–75.

[8] Andreas Binzenhöfer and Kenji Leibnitz, *Estimating Churn in Structured P2P Networks*, Managing Traffic Performance in Converged Networks (Lorne Mason, Tadeusz Drwiega, and James Yan, eds.), Lecture Notes in Computer Science, vol. 4516, Springer Berlin Heidelberg, 2007, pp. 630–641.

[9] *BitTorrent*, http://www.bittorrent.com/, last visited: January 2012.

[10] *BitTorrent PeerExchange Conventions - TheoryOrg*, http://wiki.theory.org/BiTorrentPeerExchangeConventions, last visited: January 2012.

[11] Burton H. Bloom, *Space/Time Trade-offs in Hash Coding with Allowable Errors*, Communications of the ACM **13** (1970), no. 7, 422–426.

[12] Thomas Bocek, Yehia El-khatib, Fabio V. Hecht, David Hausheer, and Burkhard Stiller, *Demonstration of the CompactPSH Incentive Scheme in a Peer-to-Peer Streaming Application*, `http://www.ieeelcn.org/prior/LCN34/lcn34demos/lcn-demo2009_bocek.pdf`, last visited: May 2012.

[13] ———, *CompactPSH: An Efficient Transitive TFT Incentive Scheme for Peer-to-peer Networks*, 34th IEEE Conference on Local Computer Networks (LCN 2009) (Zurich, Switzerland), Oct. 2009, pp. 483–490.

[14] Thomas Bocek, Wang Kun, Fabio V. Hecht, David Hausheer, and Burkhard Stiller, *PSH: A Private and Shared History-based Incentive Mechanism*, 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS 2008) (Bremen, Germany), Jul. 2008, pp. 15–27.

[15] Youmna Borghol, Sebastien Ardon, Niklas Carlsson, and Anirban Mahanti, *Toward Efficient On-Demand Streaming with BitTorrent*, NETWORKING 2010 (Mark Crovella, Laura Feeney, Dan Rubenstein, and S. Raghavan, eds.), Lecture Notes in Computer Science, vol. 6091, Springer Berlin / Heidelberg, 2010, pp. 53–66.

[16] United States District Court Northern District Of California, *Metallica vs. Napster, Inc.*, `http://news.findlaw.com/hdocs/docs/napster/napster-md030601ord.pdf`, last visited: January 2012.

[17] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh, *SplitStream: High-bandwidth Multicast in Cooperative Environments*, SIGOPS Oper. Syst. Rev. **37** (2003), no. 5, 298–313.

[18] Meeyoung Cha, Pablo Rodriguez, Jon Crowcroft, Sue Moon, and Xavier Amatriain, *Watching Television over an IP Network*, 8th ACM SIGCOMM Conference on Internet Measurement (IMC '08) (Vouliagmeni, Greece), Oct. 2008, pp. 71–84.

[19] Bin Cheng, Hai Jin, and Xiaofei Liao, *Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays*, 2007 IEEE International Conference on Communications (ICC '07) (Glasgow, Scotland), Jun. 2007, pp. 1698–1703.

[20] Bin Cheng, Lex Stein, Hai Jin, Xiaofei Liao, and Zheng Zhang, *GridCast: Improving Peer Sharing for P2P VoD*, ACM Trans. Multimedia Comput. Commun. Appl. **4** (2008), 26:1–26:31.

[21] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang, *A Case for End System Multicast (Keynote Address)*, ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '00) (Santa Clara, CA, USA), ACM, Jun. 2000, pp. 1–12.

[22] *Cisco Visual Networking Index – Forecast and Methodology, 2009–2014*, `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`, Jun. 2008.

[23] Bram Cohen, *Incentives Build Robustness in BitTorrent*, 1st Workshop on Economics of Peer-to-peer Systems (P2PECON) (Berkeley, CA, USA), Jun. 2003.

[24] The Nielsen Company, *How DVRs Are Changing the Television Landscape*, `http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/04/dvr_tvlandscape_043009.pdf`, Apr. 2009.

[25] ———, *What Consumers Watch: Nielsen's Q1 2010 Three Screen Report*, `http://www.nielsen.com/content/dam/corporate/us/en/reports-downloads/3%20Screen/2010/Three%20Screen%20Report%20(Q1%202010).pdf`, Jun. 2010.

[26] Scott A Crosby and Dan S. Wallach, *An Analysis of BitTorrent's Two Kademlia-Based DHTs*, Tech. Report TR-07-04, Department of Computer Science, Rice University, Jun. 2007.

[27] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, *Wide-area Cooperative Storage with CFS*, 18th ACM Symposium on Operating Systems Principles (SOSP '01) (Banff, Canada), Oct. 2001, pp. 202–215.

[28] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, *BASS: BitTorrent Assisted Streaming System for Video-on-Demand*, IEEE 7th Workshop on Multimedia Signal Processing (Shanghai, China), Nov. 2005, pp. 1 –4.

[29] S.E. Deering and D.R. Cheriton, *Host groups: A multicast extension to the internet protocol*, RFC 966, December 1985, Obsoleted by RFC 988.

[30] Stephen E. Deering and David R. Cheriton, *Multicast Routing in Datagram Internetworks and Extended LANs*, ACM Trans. Comput. Syst. **8** (1990), no. 2, 85–110.

[31] S. Deshpande and J. Noh, *P2TSS: Time-Shifted and Live Streaming of Video in Peer-to-peer Systems*, 2008 IEEE International Conference on Multimedia and Expo (ICME 2008) (Hannover, Germany), Jun. 2008, pp. 649 – 652.

[32] C. Diot, B.N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, *Deployment Issues for the IP Multicast Service and Architecture*, Network, IEEE **14** (2000), no. 1, 78 –88.

[33] *Distributed Tracker - Tribler*, `http://www.tribler.org/trac/wiki/DistributedTracker`, last visited: January 2012.

[34] T.T. Do, K.A. Hua, and M.A. Tantaoui, *P2VoD: Providing Fault Tolerant Video-on-demand Streaming in Peer-to-peer Environment*, IEEE International Conference on Communications (ICC 04) (Paris, France), vol. 3, Jun. 2004, pp. 1467 – 1472.

[35] Michal Feldman, Christos H. Papadimitriou, John Chuang, and Ion Stoica, *Free-riding and Whitewashing in Peer-to-peer Systems*, IEEE Journal on Selected Areas in Communications **24** (2006), no. 5, 1010–1019.

[36] M. Fiedler, T. Hoßfeld, and Phuoc Tran-Gia, *A Generic Quantitative Relationship Between Quality of Experience and Quality of Service*, Network, IEEE **24** (2010), no. 2, 36 –41.

[37] Diego Gallo, Charles Miers, Vlad Coroama, Tereza Carvalho, Victor Souza, and Per Karlsson, *A Multimedia Delivery Architecture for IPTV with P2P-Based Time-Shift Support*, 6th IEEE Conference on Consumer Communications and Networking Conference (CCCNC 2009) (Piscataway, NJ, USA), Jan. 2009, pp. 447–448.

[38] *Global Napster Usage Plummets, But New File-Sharing Alternatives Gaining Ground*, Press Release: Jupiter Media Metrix, Jul. 2001, `http://www.comscore.com/press/release.asp?id=249`.

[39] *Gnutella for Users*, `http://gnufu.net/`, last visited: January 2012.

[40] V.K. Goyal, *Multiple Description Coding: Compression Meets the Network*, Signal Processing Magazine, IEEE **18** (2001), no. 5, 74 –93.

[41] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble, *King: Estimating Latency Between Arbitrary Internet End Hosts*, 2nd ACM SIGCOMM Workshop on Internet Measurment (IMW '02) (New York, NY, USA), November 2002, pp. 5–18.

[42] Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley, *P2Cast: Peer-to-peer Patching Scheme for VoD Service*, 12th International Conference on World Wide Web (WWW'03) (Budapest, Hungary), ACM, 2003, pp. 301–309.

[43] A. Habib and S. Goose, *CommunityPVR: A Service to Deliver the Long Tail for On-Demand TV*, Multimedia, IEEE **16** (2009), no. 3, 28 –38.

[44] Anwar Al Hamra, Arnaud Legout, and Chadi Barakat, *Understanding the Properties of the BitTorrent Overlay*, CoRR **abs/0707.1820** (2007).

[45] Fabio V. Hecht, Thomas Bocek, Richard G. Clegg, Raul Landa, David Hausheer, and Burkhard Stiller, *LiveShift: Mesh-Pull P2P Live and Time-Shifted Video Streaming*, Tech. Report IFI-2010-0009, University of Zurich, Department of Informatics, Sep. 2010.

[46] ———, *LiveShift: Mesh-Pull P2P Live and Time-Shifted Video Streaming*, 36th IEEE Conference on Local Computer Networks (LCN 2011) (Bonn, Germany), October 2011, pp. 319–327.

[47] Fabio V. Hecht, Thomas Bocek, and David Hausheer, *The Pirate Bay 2008-12 Dataset*, `http://www.csg.uzh.ch/publications/data/piratebay/`, last visited: January 2012.

[48] Fabio V. Hecht, Thomas Bocek, Cristian Morariu, David Hausheer, and Burkhard Stiller, *LiveShift: Peer-to-peer Live Streaming with Distributed Time-Shifting*, 8th International Conference on Peer-to-peer Computing (P2P'08) (Aachen, Germany), Sep. 2008, pp. 187–188.

[49] Fabio V. Hecht, Thomas Bocek, Flávio Roberto Santos, and Burkhard Stiller, *Playback Policies for Live and On-Demand P2P Video Streaming*, IFIP Networking (Prague, Czech Republic), May 2012, pp. 15–28.

[50] Fabio V. Hecht, Thomas Bocek, and Burkhard Stiller, *B-Tracker: Improving Load Balancing and Efficiency in Distributed P2P Trackers*, 11th IEEE International Conference on Peer-to-peer Computing (P2P'11) (Kyoto, Japan), Sep. 2011, pp. 310–313.

[51] ———, *B-Tracker: Improving Load Balancing and Efficiency in Distributed P2P Trackers*, Tech. Report IFI-2011-0003, University of Zurich, Department of Informatics, Apr. 2011.

[52] Oliver Heckmann, Axel Bock, Andreas Mauthe, and Ralf Steinmetz, *The eDonkey File-Sharing Network*, Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications **51** (2004), 2–6.

[53] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and K.W. Ross, *A Measurement Study of a Large-Scale P2P IPTV System*, IEEE Transactions on Multimedia **9** (2007), no. 8, 1672–1687.

[54] Xiaojun Hei, Yong Liu, and K. W. Ross, *IPTV over P2P Streaming Networks: The Mesh-Pull Approach*, Communications Magazine, IEEE **46** (2008), no. 2, 86–92.

[55] T. Hoßfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen, *Initial Delay vs. Interruptions: Between the Devil and the Deep Blue Sea*, Fourth International Workshop on Quality of Multimedia Experience (QoMEX) (Yarra Valley, Australia), Jul. 2012, pp. 1–6.

[56] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, Phuoc Tran-Gia, and R. Schatz, *Quantification of YouTube QoE via Crowdsourcing*, IEEE International Symposium on Multimedia (ISM 2011) (Dana Point, CA, USA), Dec. 2011, pp. 494–499.

[57] Tobias Hoßfeld, Sebastian Biedermann, Raimund Schatz, Alexander Platzer, Sebastian Egger, and Markus Fiedler, *The Memory Effect and Its Implications on Web QoE Modeling*, 23rd International Teletraffic Congress (ITC '11) (San Francisco, CA, USA), Sep. 2011, pp. 103–110.

[58] Tobias Hoßfeld, David Hausheer, Fabio Hecht, Frank Lehrieder, Simon Oechsner, Ioanna Papafili, Peter Racz, Sergios Soursos, Dirk Staehle, George D. Stamoulis, Phuoc Tran-Gia, and Burkhard Stiller, *An Economic Traffic Management Approach to Enable the TripleWin for Users, ISPs, and Overlay Providers*, FIA Prague Book, IOS Press Books Online, Towards the Future Internet - A European Research Perspective, 5 2009, pp. 24–34.

[59] Hung-Chang Hsiao and Che-Wei Chang, *A Symmetric Load Balancing Algorithm with Performance Guarantees for Distributed Hash Tables*, IEEE Transactions on Computers **62** (2013), no. 4, 662–675.

[60] Yan Huang, Tom Z.J. Fu, Dah-Ming Chiu, John C.S. Lui, and Cheng Huang, *Challenges, Design and Analysis of a Large-Scale P2P-VoD System*, SIGCOMM Comput. Commun. Rev. **38** (2008), no. 4, 375–388.

[61] Q. Huynh-Thu and M. Ghanbari, *Scope of Validity of PSNR in Image/Video Quality Assessment*, Electronics Letters **44** (2008), no. 13, 800 –801.

[62] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr., *Overcast: Reliable Multicasting with on Overlay Network*, 4th Symposium on Operating System Design & Implementation (OSDI'00) (San Diego, CA, USA), vol. 4, USENIX Association, Oct. 2000, pp. 197–212.

[63] *Java.com*, `http://java.com/en/`, last visited: May 2012.

[64] Xiaoke Jiang, Jun Bi, Yangyang Wang, Zhijie He, Wei Zhang, and Hongcheng Tian, *IPv6 Evolution, Stability and Deployment*, 19th annual IEEE International Conference on Network Protocols (ICNP 2011) (Vancouver, Canada), Oct. 2011, pp. 123–124.

[65] Eunsam Kim and J.C.L. Liu, *Design of HD-quality Streaming Networks for Real-time Content Distribution*, IEEE Transactions on Consumer Electronics **52** (2006), no. 2, 392–401.

[66] R. Kumar, Yong Liu, and K. Ross, *Stochastic Fluid Theory for P2P Streaming Systems*, 26th IEEE International Conference on Computer Communications (INFOCOM 2007) (Anchorage, AL, USA), May 2007, pp. 919 –927.

[67] Raul Landa, David Griffin, Richard G. Clegg, Eleni Mykoniati, and Miguel Rio, *A Sybilproof Indirect Reciprocity Mechanism for Peer-to-peer Networks*, 28th IEEE International Conference on Computer Communications (INFOCOM 2009) (Rio de Janeiro, Brazil), Apr. 2009, pp. 343–351.

[68] Seung-Bum Lee, G.-M. Muntean, and A.F. Smeaton, *Performance-Aware Replication of Distributed Pre-Recorded IPTV Content*, IEEE Transactions on Broadcasting **55** (2009), no. 2, 516 –526.

[69] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi, *Rarest First and Choke Algorithms are Enough*, 6th ACM SIGCOMM Conference on Internet Measurement (IMC '06) (Rio de Janeiro, Brazil), ACM, Oct. 2006, pp. 203–216.

[70] Frank Lehrieder, Simon Oechsner, Tobias Hoßfeld, Zoran Despotovic, Wolfgang Kellerer, and Maximilian Michel, *Can P2P-Users Benefit from*

Locality-Awareness?, IEEE Tenth International Conference on Peer-to-Peer Computing (P2P 2010) (Delft, The Netherlands), Aug. 2010, pp. 1–9.

[71] Bo Li, Susu Xie, Yang Qu, G.Y. Keung, Chuang Lin, Jiangchuan Liu, and Xinyan Zhang, *Inside the New Coolstreaming: Principles, Measurements and Performance Implications*, 27th Conference on Computer Communications (INFOCOM 2008) (Phoenix, AZ, USA), Apr. 2008, pp. 1031 –1039.

[72] Chao Liang, Yang Guo, and Yong Liu, *Is Random Scheduling Sufficient in P2P Video Streaming?*, 28th International Conference on Distributed Computing Systems (ICDCS '08) (Beijing, China), Jun. 2008, pp. 53 –60.

[73] Jian Liang, Rakesh Kumar, and Keith W. Ross, *The FastTrack Overlay: A Measurement Study*, Computer Networks – Overlay Distribution Structures and their Applications **50** (2006), no. 6, 842 – 858.

[74] Y. Liu and G. Simon, *Peer-to-peer Time-shifted Streaming Systems*, ArXiv e-prints (2009).

[75] Yaning Liu and Gwendal Simon, *Distributed Delivery System for Time-Shifted Streaming Systems*, 2010 IEEE 35th Conference on Local Computer Networks (LCN 2010), October 2010, pp. 276 –279.

[76] ———, *Peer-Assisted Time-shifted Streaming Systems: Design and Promises*, 2011 IEEE International Conference on Communications (ICC 2011) (Kyoto,Japan), June 2011, pp. 1–5.

[77] Yong Liu, Yang Guo, and Chao Liang, *A Survey on Peer-to-peer Video Streaming Systems*, Peer-to-peer Networking and Applications **1** (2008), no. 1, 18–28.

[78] *LiveShift: P2P Live Video Streaming with Time Shifting Ability*, `https://github.com/zegotinha/LiveShift/`, last visited: October 2012.

[79] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer, *Free Riding in BitTorrent is Cheap*, 5th Workshop on Hot Topics in Networks (HotNets) (Irvine, CA, USA), Nov. 2006, pp. 85–90.

[80] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim, *A Survey and Comparison of Peer-to-peer Overlay Network Schemes*, IEEE Communications Surveys and Tutorials **7** (2005), 72–93.

[81] Nazanin Magharei and Reza Rejaie, *Understanding Mesh-based Peer-to-peer Streaming*, 2006 International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '06) (Newport, RI,

USA), ACM, Nov. 2006, pp. 10:1–10:6.

[82] Nazanin Magharei and Reza Rejaie, *Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches*, 26th IEEE International Conference on Computer Communications (INFOCOM 2007) (Anchorage, AL, USA), May 2007, pp. 1424–1432.

[83] Nazanin Magharei and Reza Rejaie, *PRIME: Peer-to-peer Receiver-Driven Mesh-Based Streaming*, IEEE/ACM Transactions on Networking **17** (2009), 1052–1065.

[84] P. Maymounkov and D. Mazieres, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*, 1st International Workshop on Peer-to-peer Systems (IPTPS 2002) (London, UK), Mar. 2002, pp. 53–65.

[85] Nelson Minar, Marc Hedlund, Clay Shirky, Tim O'Reilly, Dan Bricklin, David Anderson, Jeremie Miller, Adam Langley, Gene Kan, Alan Brown, Marc Waldman, Lorrie Faith Cranor, Aviel Rubin, Roger Dingledine, Michael Freedman, David Molnar, Rael Dornfest, Dan Brickley, Theodore Hong, Richard Lethin, Jon Udell, Nimisha Asthagiri, Walter Tuvell, and Brandon Wiley, *Peer-to-peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[86] J. J. D. Mol, *Free-riding, Resilient Video Streaming in Peer-to-peer Networks*, Ph.D. thesis, Delft University of Technology, January 2010.

[87] J. J. D. Mol, A. Bakker, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, *The Design and Deployment of a BitTorrent Live Video Streaming Solution*, Intl. Symposium on Multimedia (Los Alamitos, CA, USA), Dec. 2009, pp. 342–349.

[88] Jeonghun Noh, Aditya Mavlankar, Pierpaolo Baccichet, and Bernd Girod, *Time-Shifted Streaming in a Peer-to-peer Video Multicast System*, 28th IEEE Conference on Global Telecommunications (GLOBECOM '09) (Honolulu, HI, USA), IEEE Press, Dec. 2009, pp. 6025–6030.

[89] Zhipeng Ouyang, Lisong Xu, and B. Ramamurthy, *Providing NPR-Style Time-Shifted Streaming in P2P Systems*, 2011 20th International Conference on Computer Communications and Networks (ICCCN '11), Aug. 2011, pp. 544–549.

[90] Vassilis G. Papanicolaou, George E. Kokolakis, and Shahar Boneh, *Asymptotics for the Random Coupon Collector Problem*, Journal of Computational

and Applied Mathematics **93** (1998), no. 2, 95 – 105.

[91] A. Pathan and R. Buyya, *A Taxonomy and Survey of Content Delivery Networks*, Technical Report, GRIDS-TR-2007-4, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Feb. 2007.

[92] *Peer Exchange*, `http://wiki.vuze.com/w/Peer_Exchange`, last visited: January 2012.

[93] J. Peterson, V. Gurbani, and E. Marocco, *ALTO Working Group Charter*, `http://www.ietf.org/html.charters/alto-charter.html`, last visited: October 2012.

[94] Michael Piatek, Arvind Krishnamurthy, Arun Venkataramani, Richard Yang, David Zhang, and Alexander Jaffe, *Contracts: Practical Contribution Incentives for P2P Live Streaming*, 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10) (Berkeley, CA, USA), April 2010, pp. 377–391.

[95] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips, *Tribler: A Social-based Peer-to-peer System*, Concurrency and Computation: Practice and Experience **20** (2008), 127–138.

[96] *PPLive*, `http://www.pplive.com/`, last visited: January 2012.

[97] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica, *Load Balancing in Structured P2P Systems*, Peer-to-Peer Systems II (M.Frans Kaashoek and Ion Stoica, eds.), Lecture Notes in Computer Science, vol. 2735, Springer Berlin Heidelberg, 2003, pp. 68–79.

[98] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz, *Handling Churn in a DHT*, Annual Conference on USENIX Annual Technical Conference (ATEC '04) (Boston, MA), USENIX Association, Jul. 2004, pp. 127–140.

[99] Antony Rowstron and Peter Druschel, *Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-peer Systems*, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) (Heidelberg, Germany), Nov. 2001, pp. 329–350.

[100] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy, *An Analysis of Internet Content Delivery Systems*,

SIGOPS Oper. Syst. Rev. **36** (2002), 315–327.

[101] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, *A Measurement Study of Peer-to-peer File Sharing Systems*, SIGCOMM Computer Communication Review **32** (2002), no. 1, 82–82.

[102] Hendrik Schulze and Klaus Mochalski, *Internet Study 2008/2009*, `http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf`, last visited: January 2012.

[103] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari, *Will IPTV Ride the Peer-to-peer Stream?*, IEEE Communications Magazine **45** (2007), no. 6, 86–92.

[104] *SETI@home*, `http://setiathome.berkeley.edu/`, last visited: January 2012.

[105] H. Shen and C.-Z. Xu, *Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks*, IEEE Transactions on Parallel and Distributed Systems **18** (2007), no. 6, 849 –862.

[106] S. Shinohara, Y. Akematsu, and M. Tsuji, *Panel Data Analysis of Factors of Broadband Services Diffusion in OECD Countries: Focus on Deployment and Migration*, 22nd European Regional ITS Conference, Budapest 2011: Innovative ICT Applications - Emerging Regulatory, Economic and Policy Issues 52157, International Telecommunications Society (ITS), 2011.

[107] *SopCast - Free P2P Internet TV*, `http://www.sopcast.org/`, last visited: January 2012.

[108] R. Stankiewicz, P. Cholda, and A. Jajszczyk, *QoX: What is it really?*, Communications Magazine, IEEE **49** (2011), no. 4, 148 –158.

[109] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack, *A Global View of KAD*, 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07) (San Diego, CA, USA), ACM, Oct. 2007, pp. 117–122.

[110] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Aug. 2001, pp. 149–160.

[111] A. Takeda, T. Oide, and A. Takahashi, *New Structured P2P Network with Dynamic Load Balancing Scheme*, 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA), Mar. 2011, pp. 108–113.

[112] Andrew Tanenbaum, *Computer Networks*, 4th ed., Prentice Hall Professional Technical Reference, 2002.

[113] *Testbed Infrastructure for Research Activities – CSG*, `http://www.csg.uzh.ch/services/testbed/`, last visited: January 2012.

[114] *TiVo Premiere: Much More than a DVR - TiVo*, `http://www.tivo.com/`, last visited: January 2012.

[115] *TomP2P, a P2P-Based Key-Value Pair Storage Library*, `http://tomp2p.net/`, last visited: January 2012.

[116] Kurt Tutschku, *A Measurement-Based Traffic Profile of the eDonkey Filesharing Service*, Passive and Active Network Measurement (Chadi Barakat and Ian Pratt, eds.), Lecture Notes in Computer Science, vol. 3015, Springer Berlin / Heidelberg, 2004, pp. 12–21.

[117] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen, *A Survey of DHT Security Techniques*, ACM Comput. Surv. **43** (2011), 8:1–8:49.

[118] A. Vlavianos, M. Iliofotou, and M. Faloutsos, *BiToS: Enhancing BitTorrent for Supporting Streaming Applications*, 25th IEEE International Conference on Computer Communications (INFOCOM 2006) (Barcelona, Spain), Apr. 2006, pp. 1–6.

[119] *vlcj - Java Framework for the vlc Media Player*, `http://code.google.com/p/vlcj/`, last visited: January 2012.

[120] R. Winter and M. Sridhavan, *Low Extra Delay Background Transport (ledbat) Working Group Charter*, `http://tools.ietf.org/wg/ledbat/charters`, last visited: October 2012.

[121] Di Wu, Ye Tian, and Kam-Wing Ng, *Resilient and Efficient Load Balancing in Distributed Hash Tables*, Journal of Network and Computer Applications **32** (2009), no. 1, 45 – 60.

[122] W.-P.K. Yiu, Xing Jin, and S.-H.G. Chan, *VMesh: Distributed Segment Storage for Peer-to-Peer Interactive Video Streaming*, IEEE Journal on Selected Areas in Communications **25** (2007), no. 9, 1717–1731.

[123] *YouTube - Broadcast Yourself*, `http://www.youtube.com/`, last visited: January 2012.

[124] Chao Zhang, P. Dhungel, Di Wu, and K.W. Ross, *Unraveling the BitTorrent Ecosystem*, IEEE Transactions on Parallel and Distributed Systems **22** (2011), no. 7, 1164 –1177.

[125] Meng Zhang, Jian-Guang Luo, Li Zhao, and Shi-Qiang Yang, *A Peer-to-peer Network for Live Media Streaming Using a Push-pull Approach*, 13th Annual ACM International Conference on Multimedia (MULTIMEDIA '05) (Singapore, Singapore), ACM, 2005, pp. 287–290.

Other Author Publications

Conference, Workshop, and Demonstration Papers

- T. Bocek, D. Peric, F. V. Hecht, D. Hausheer, B. Stiller, *PeerVote: A Decentralized Voting Mechanism for P2P Collaboration Systems,* 3rd ACM/IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS 2009), Springer, July 2009.
- T. Bocek, F. V. Hecht, D. Hausheer, D. Peric, B. Stiller, *Incentives for Voting-based Quality Control and Document Storage in P2P Collaboration Systems,* 18th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2009), Verlags, pages 1-2, June 2009.
- T. Bocek, F. V. Hecht, D. Hausheer, E. Hunt, B. Stiller, *Mobile P2P Fast Similarity Search,* 6th Annual IEEE Consumer Communications & Networking Conference (CCNC 2009), January 2009.
- T. Bocek, F. V. Hecht, E. Hunt, D. Hausheer, B. Stiller, *Fast Similarity Search for Structured P2P Systems,* 1st EMANICS Workshop on Peer-to-Peer Management, pages 1-23, March 2008.
- F. V. Hecht, B. Stiller, *Report- and Reciprocity-based Incentive Mechanisms for Live and On-demand P2P Video Streaming,* Proceedings of AIMS 2010, Springer LNCS 6155, June 2010.
- F. V. Hecht, B. Stiller, *Economic Traffic Management: Mechanisms and Applications,* Telecommunication Economics, Lecture Notes in Computer Science (LNCS), ISBN: 978-3-642-30381-4, Springer, Heidelberg, Germany, Vol. 7216, pages 188-198, May 2012.
- F. V. Hecht, *Enabling Next Generation Peer-to-peer Services,* 2nd International Conference on Autonomous Infrastructure, Management and Security Resilient Networks and Services (AIMS), July 2008.
- F. V. Hecht, *Improving the Performance of P2P Streaming through an Overlay-operator Interface like ALTO/SIS,* 2nd EMANICS Workshop on Peer-to-Peer Management, April 2009.

- F. V. Hecht, T. Bocek, N. Bär, R. Erdin, B. Kuster, M. Zeeshan, B. Stiller, Radiommender: P2P On-line Radio with a Distributed Recommender System. Twelfth IEEE International Conference on Peer-to-Peer Computing (IEEE P2P'12), IEEE Computer Society, September 2012.
- M. V. J. Heikkinen, T. Casey, Fabio Hecht, *Value Analysis of Centralized and Distributed Communications and Video Streaming,* The Journal of Policy, Regulation and Strategy for Telecommunications, Information and Media, ISSN 1463-6697, Emerald Group Publishing Limited, Emerald Group Publishing Limited, Vol. 12, No. 5, pages 42-58, August 2010.
- T. Hoßfeld, D. Hausheer, F. V. Hecht, D. Peric, S. Oechsner, I. Papafili, P. Racz, S. Soursos, D. Staehle, G. D. Stamoulis, P. Tran-Gia, B. Stiller, *An Economic Traffic Management Approach to Enable the TripleWin for Users, ISPs, and Overlay Providers,* Towards the Future Internet - A European Research Perspective, ISBN: 978-1-60750-007-0, IOS Press, pages 24-34, May 2009.
- D. J. Lutz, D. Lamp, P. Mandic, F. V. Hecht, B. Stiller, *Charging of SAML-based Federated VoIP Services,* 5th International Conference for Internet Technology and Secured Transactions (ICITST 2010), IEEE, Vol. 5, November 2010.
- G. S. Machado, F. V. Hecht, M. Waldburger, B. Stiller, *Bypassing Cloud Providers Data Validation To Store Arbitrary Data,* IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 2013.
- D. Peric, T. Bocek, F. V. Hecht, D. Hausheer, B. Stiller, *Brief Announcement: The Design and Evaluation of a Distributed Reliable File System,* 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2009), Springer LNCS, Vol. 5873, pages 797-798, November 2009.
- D. Peric, T. Bocek, F. V. Hecht, D. Hausheer, B. Stiller, *The Design and Evaluation of a Distributed Reliable File System,* Second International Workshop on Reliability, Availability, and Security (WRAS 2009), IEEE Computer Society, December 2009.
- P. Racz, S. Soursos, M. Á. C. Rodriguez, S. Spirou, F. V. Hecht, I. Papafili, G. D. Stamoulis, H. Hasan, B. Stiller, *Economic Traffic Management for Overlay Networks,* ICT-MobileSummit 2009, IIMC International Information Management Corporation, June 2009.
- M. Á. C. Rodríguez, J. A. S. García, A. M. Martín-Carnerero, P. Racz, F. V. Hecht, S. Spirou, I. Papafili, G. Stamoulis, W. Kellerer, K. Wajda, *NGN Us-*

age in Future Internet Scenarios, Future Network and MobileSummit 2010, IIMC International Information Management Corporation, June 2010.

- F. R. Santos, W. L. da Costa Cordeiro, M. Barcellos, L. Paschoal Gaspary, F. V. Hecht, B. Stiller, *Slowing Down to Speed Up: Mitigating Collusion Attacks in Content Distribution Systems,* IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 2013.

Technical Reports

- T. Bocek, D. Peric, F. V. Hecht, D. Hausheer, B. Stiller, *Towards a Decentralized Voting Mechanism for P2P Collaboration Systems,* IFI Technical Report, March 2009.
- D. Hausheer, T. Bocek, F. V. Hecht, C. Morariu, G. Schaffrath, B. Stiller (Eds.), *P2P Challenge Task 2008,* IFI Technical Report, July 2008.
- B. Stiller, K. Farkas, F. V. Hecht, G. Machado, A. Vancea, M. Waldburger (Eds.), Communication Systems V. IFI Technical Report, IFI 2012.06, pages 1-89, August 2012.
- B. Stiller, K. Farkas, F. V. Hecht, G. Machado, P. Poullie, F. Santos, C. Tsiaras, A. Vancea, M. Waldburger, Internet Economics VI. IFI Technical Report, No. IFI-2012.02, pages 1-147, April 2012.
- B. Stiller, F. V. Hecht, M. Lo Bosco, P. Racz, G. S. Machado, A. Vancea (Eds.), *Internet Economics V,* IFI Technical Report, July 2010.
- B. Stiller, Hasan, F. V. Hecht, G. Machado, A. Vancea, M. Waldburger (Eds.), *Mobile Systems V*. IFI Technical Report, No. IFI-2011.05, pages 1-100, November 2011.
- B. Stiller, Hasan, F. V. Hecht, Guilherme Machado, A. Vancea, M. Waldburger, *Communication Systems IV,* IFI Technical Report, pages 1-81, January 2011.
- B. Stiller, T. Bocek, F. V. Hecht, C. Morariu, P. Racz, A. Vancea, M. Waldburger (Eds.), *Communication Systems III,* IFI Technical Report, June 2009.
- B. Stiller, T. Bocek, F. V. Hecht, C. Morariu, P. Racz, G. Schaffrath (Eds.), *Communication Systems II,* IFI Technical Report, No. ifi-2008.01, January 2008.
- B. Stiller, T. Bocek, F. V. Hecht, G. Machado, P. Racz, M. Waldburger (Eds.), *Mobile Systems IV,* IFI Technical Report , January 2010.

Others

- G. Hoekstra, D. Bijwaard, F. V. Hecht et al., *Report on Application of Key Concepts to Service Provisioning,* Deliverable DII-313 of the Daidalos II project, pages 133, October 2007.
- P. Racz, F. V. Hecht (Eds.), *Economic Traffic Management Systems Architecture Design (Initial Version),* Deliverable D3.1 of SmoothIT, October 2008.
- Fabio V. Hecht, Patrick Poullie, Andrei Vancea, Burkhard Stiller, Attacks on Internet Names. Readme Alumni - Das Bulletin der Alumni Wirtschaftsinformatik Universität Zürich, Alumni Witschaftsinformatik Universität Zürich, No. 28, pages 14-15, October 2012.

Appendix A

A.1 LiveShift Message Specification

This appendix displays complete message format specification of the LiveShift Protocol. The protocol itself is described in Section 3.3. All messages inherit the basic fields from Abstract Message, displayed in Table A.1. Thus, messages specified in Tables A.2-A.12 start with the fields of Abstract Message, followed by their specific ones. All sizes are represented in bytes.

Table A.1: Abstract Message

Field	*Size*	*Note*
Signature	1	0x15 = LiveShift
Protocol Version	1	1 for this protocol
Message ID	1	Wraps over, starts at 1
Reply Message ID?	1	0=absent, 1=present
Reply Message ID	1	Message ID this message is a reply for
Message Type	1	B Block Request Message C Block Reply Message D Disconnect Message G Granted Message H Have Message I Interested Message P Ping Message Q Queued Message S Peer Suggestion V Subscribed Message X Subscribe Message

Table A.2: Subscribe Message

Field	*Size*	*Note*
Segment Identifier	13	Channel ID (4 bytes) Substream (1 byte) Segment Number (8 bytes)
Peer Name Size	1	
Peer Name	Variable	
Peer Upload Capacity	4	float

Table A.3: Ping Message

Field	*Size*	*Note*
Request/reply	1	1=request 0=reply

Table A.4: Have Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Block Number	4	
Do Have?	1	1=have, 0=don't have

Table A.5: Disconnect Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Flags	1	1st least significant bit (LSB)=stop Uploading 2nd LSB=stop Downloading

Table A.6: Block Request Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Block Number	4	

Table A.7: Block Reply Message

Field	*Size*	*Note*
Block Reply Code	1	1 GRANTED, 2 DONT_HAVE, 3 REJECTED
Segment Identifier	13	As in Table A.2
Block Number	4	
Hop Count	1	
Number of Packets	4	
For each packet:		
Time	8	Milliseconds
Sequence Number	8	
Substream	1	
Video Data Length	4	
Video Data	Variable	Video Packets

Table A.8: Subscribed Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Not	1	1 = not subscribed 0 = subscribed
Timeout	4	Milliseconds
Segment Block Map	15	Only present if not=0
Peer Name Size	1	
Peer Name	Variable	Human-friendly name

Table A.9: Queued Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Not	1	1 = not subscribed 0 = subscribed
Timeout	4	Milliseconds

Table A.10: Granted Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Not	1	1 = not subscribed 0 = subscribed
Timeout	4	Milliseconds
Timeout Inactive	4	Milliseconds

Table A.11: Interested Message

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Not	1	1=not interested 0=interested

Table A.12: Peer Suggestion

Field	*Size*	*Note*
Segment Identifier	13	As in Table A.2
Number of Suggested Peers	1	
For each suggested peer:		
DHT Address	20	160-bit identifier
Socket Address Size	1	
Socket Address	9 (IPv4), 21 (IPv6)	For connecting
Peer Name Size	1	
Peer Name	Variable	Human-friendly name

Appendix B

B.1 Complete Playback Policies CDF Plots

This appendix displays complete CDF graphs at playback positions 5, 10, and 15, obtained for all investigated playback policies and parameters investigated. These results complement the ones displayed and discussed in Section 4.3.

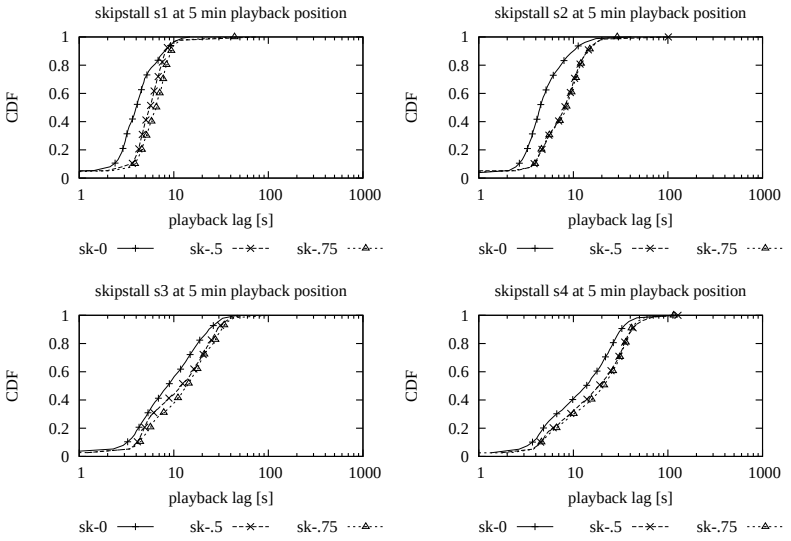

Figure B.1: CDF of playback lag under Skip/Stall playback policy at 5 min playback position

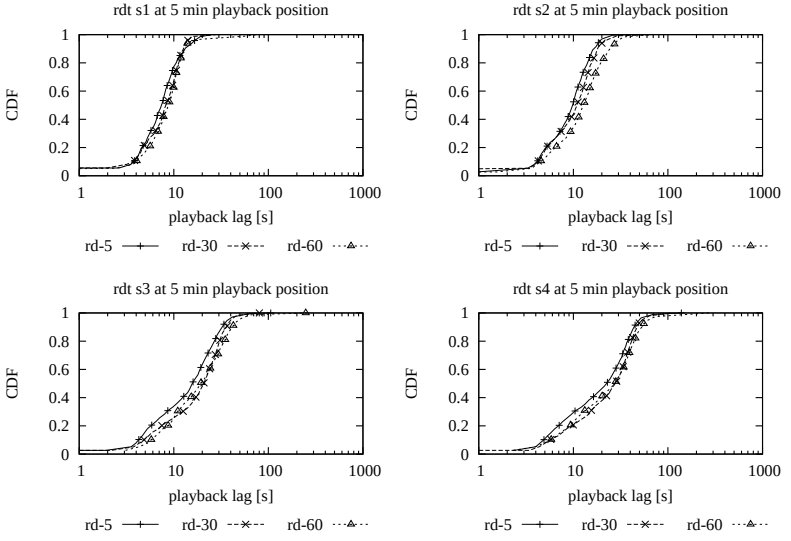

Figure B.2: CDF of playback lag under Skip/Stall playback policy at 5 min playback position

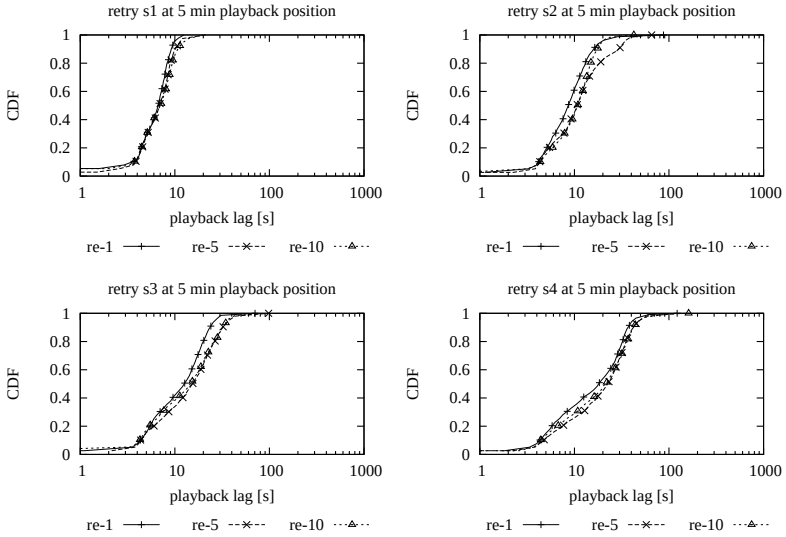

Figure B.3: CDF of playback lag under Skip/Stall playback policy at 5 min playback position

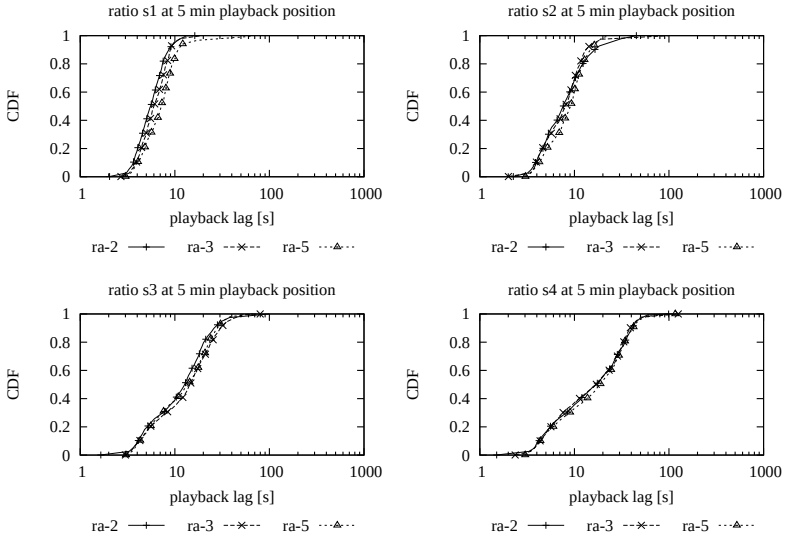

Figure B.4: CDF of playback lag under Skip/Stall playback policy at 5 min playback position

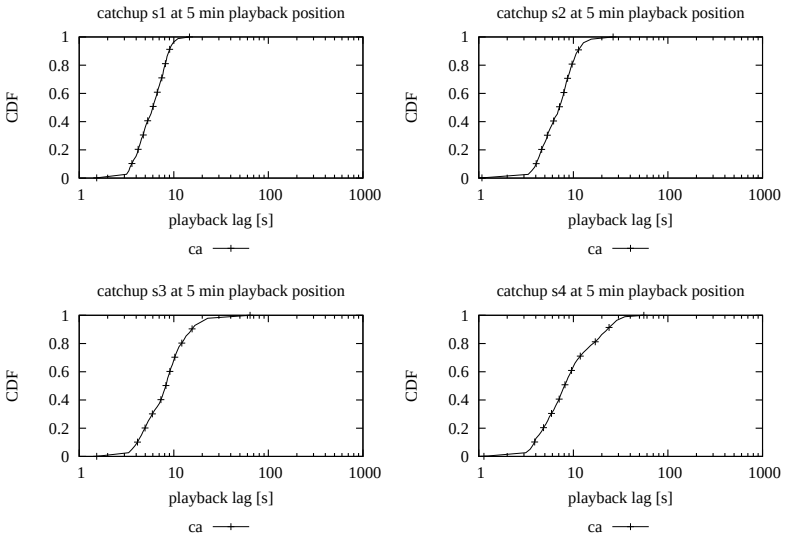

Figure B.5: CDF of playback lag under Skip/Stall playback policy at 5 min playback position

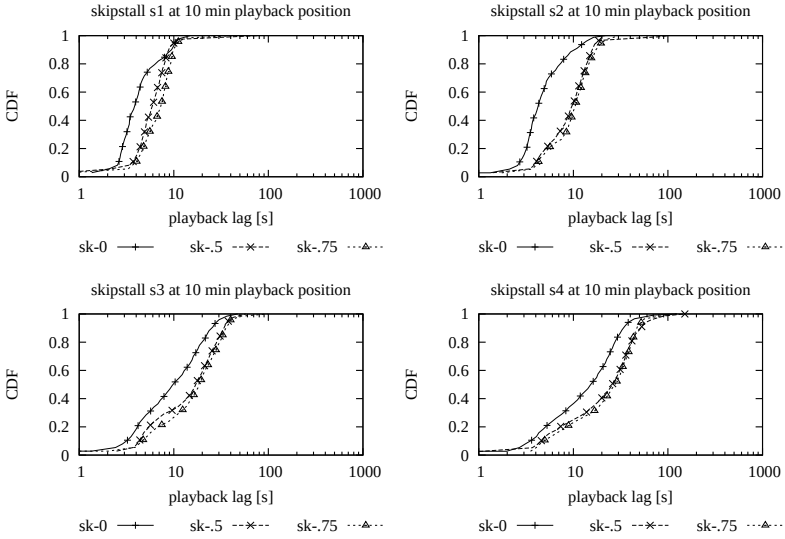

Figure B.6: CDF of playback lag under Skip/Stall playback policy at 10 min playback position

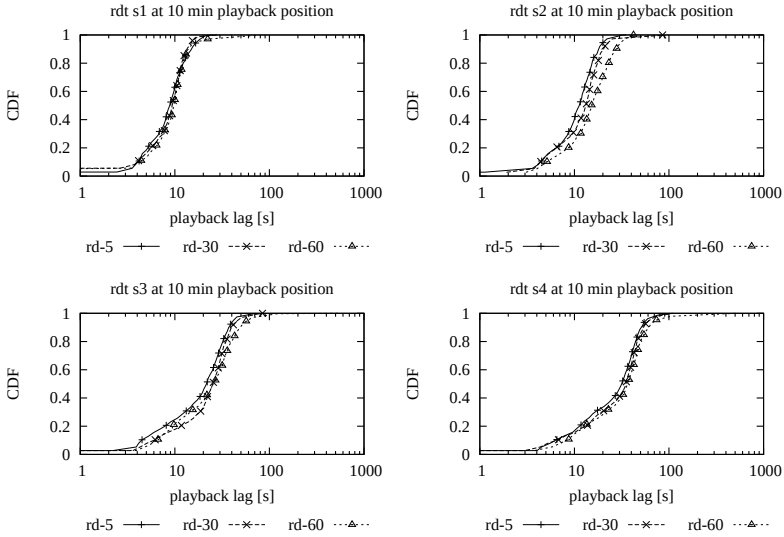

Figure B.7: CDF of playback lag under Skip/Stall playback policy at 10 min playback position

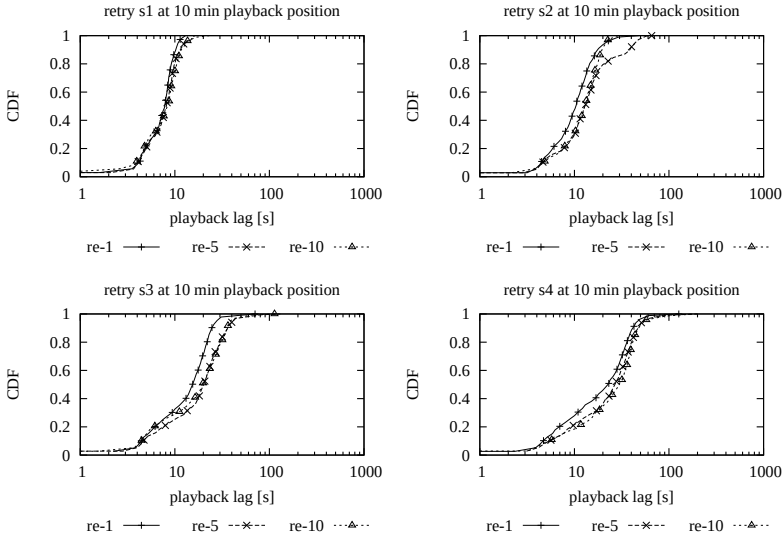

Figure B.8: CDF of playback lag under Skip/Stall playback policy at 10 min playback position

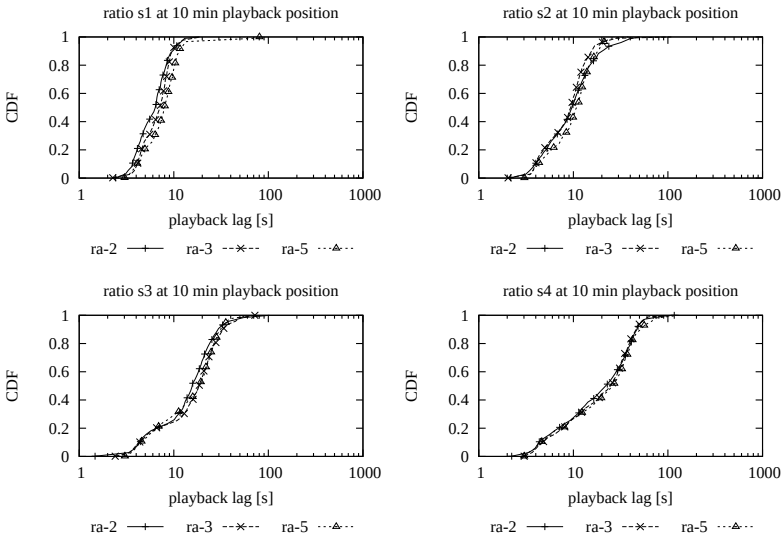

Figure B.9: CDF of playback lag under Skip/Stall playback policy at 10 min playback position

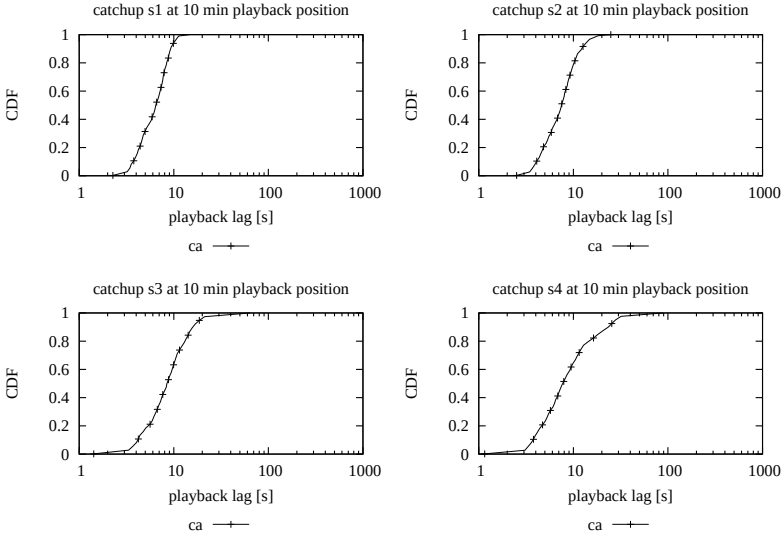

Figure B.10: CDF of playback lag under Skip/Stall playback policy at 10 min playback position

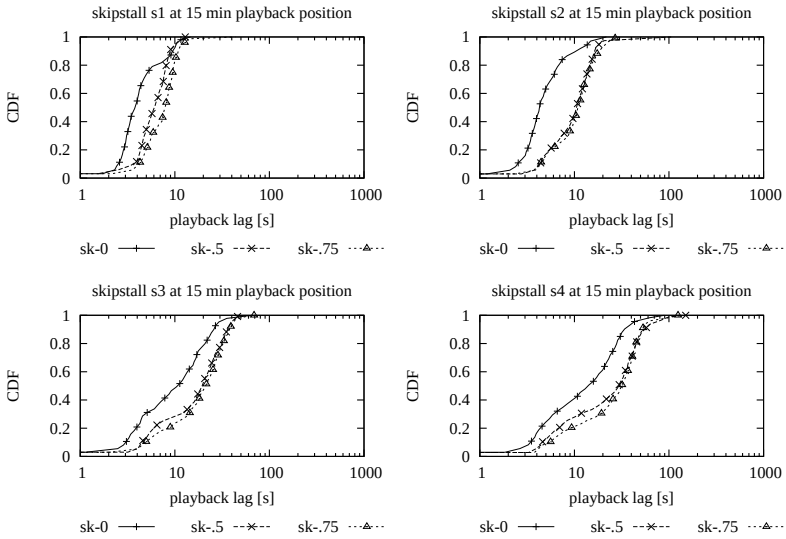

Figure B.11: CDF of playback lag under Skip/Stall playback policy at 15 min playback position

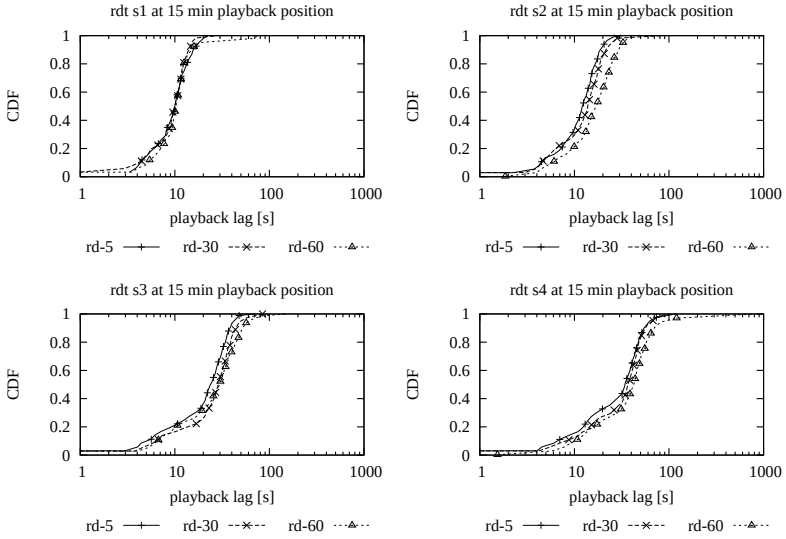

Figure B.12: CDF of playback lag under Skip/Stall playback policy at 15 min playback position

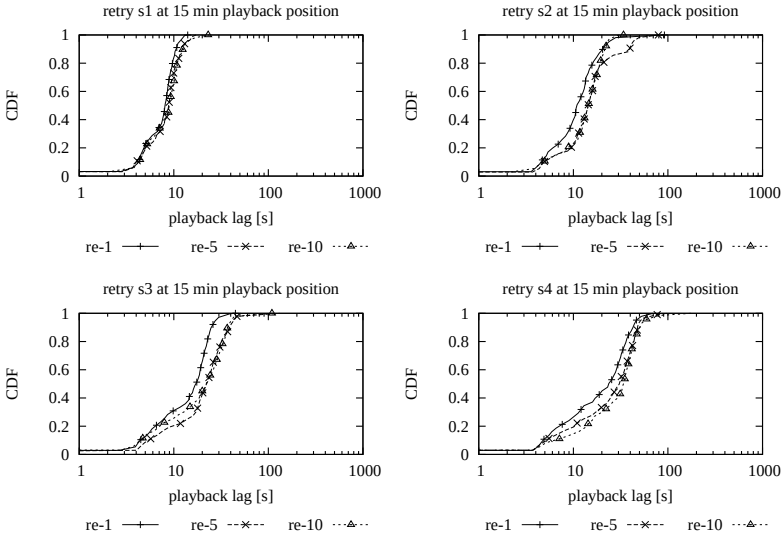

Figure B.13: CDF of playback lag under Skip/Stall playback policy at 15 min playback position

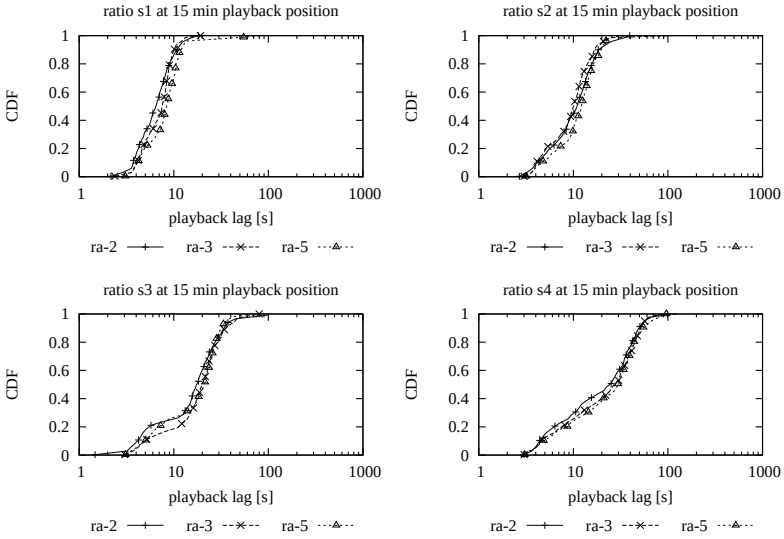

Figure B.14: CDF of playback lag under Skip/Stall playback policy at 15 min playback position

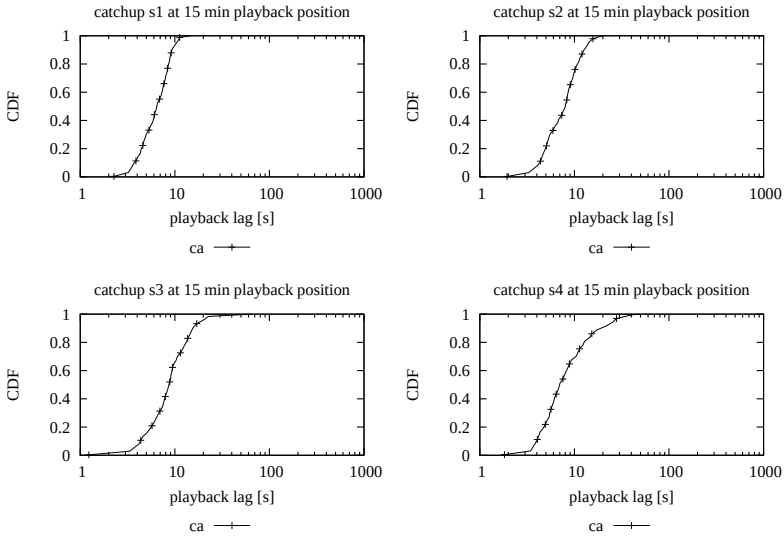

Figure B.15: CDF of playback lag under Skip/Stall playback policy at 15 min playback position

Listing of Figures

Listing of Tables

Acknowledgments

I am very grateful to a number of people that have helped me, directly or indirectly, through the arduous work that culminates with this thesis.

Firstly, I would like to thank Prof. Dr. Burkhard Stiller for believing in me, allowing me to join the Communication Systems Group (CSG) at the University of Zurich, giving me freedom to research topics that I found interesting, and precious help, feedback, and understanding during several difficult stages. In addition, I would like to thank Dr. Tobias Hoßfeld for being my co-supervisor and providing me with highly valuable feedback. Thank you!

I would also like to warmly thank "my" co-authors, in particular Dr. Thomas Bocek for co-authoring more than 10 scientific papers together, Dr. Cristian Morariu and Prof. Dr. David Hausheer for giving initial ideas and discussion about LiveShift, Dr. Raul Landa and Dr. Richard G. Clegg for fruitful discussions and hands-on support during and after my short stay in London, and Flávio Roberto Santos for important contributions in my final steps. I'm particularly grateful to Dr. Martin Waldburger for providing the German translation of this thesis' abstract. You all have greatly contributed to both my work and my overall development as a scientist. Furthermore, I would like to thank all 18 assignment, diploma, bachelor, and master students that I have supervised while working on this thesis for the rich discussions and contributions.

It has been an immense pleasure working at the CSG and enjoying the true friendship of my work colleagues, including warm discussions about diverse subjects with current colleagues Daniel Dönni, Andri Lareida, Guilherme Machado, Patrick Poullie, Christos Tsiaras, Andrei Vancea, and Dr. Martin Waldburger, as well as former colleagues Dr. Hasan, Maurizio Lo Bosco, Peter Ming, Dalibor Peric, Dr. Peter Racz, and Gregor Schaffrath.

Finally, I would like to thank my girlfriend Anna Paula de Oliveira for the unconditional support during all these years, as well as my brother, parents and family, I love you all!

Curriculum Vitae

Fabio Victora Hecht was born on December 16, 1980, in Porto Alegre, Rio Grande do Sul, Brazil. In 2004, he has obtained a five-year degree named "Bacharelado" – equivalent to a Swiss Diplom – in Computer Science from Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. His thesis was entitled "Study and Implementation of the NETCONF Protocol", in which different transport protocols for NETCONF were compared for performance and overhead. During and after his studies, Fabio has worked as a freelancer programmer and project manager, starting his own company in Porto Alegre in 2002.

But it was time for a change, and by mid-2007, Fabio Hecht has moved to Zurich, Switzerland to become a doctoral student until mid-2012, as well as a research assistant in the Communication Systems Group at the Department of Informatics of the University of Zurich. The work has involved a multitude of tasks, including distributed systems architecture and protocol development and implementation, managing server infrastructure, building automated scripts, and analyzing large amounts of data. Fabio has been involved in the following research projects: "Peer-to-Peer Live Video Streaming with Distributed Time-Shifting (LiveShift)", "SmoothIT: Simple Economic Management Approaches of Overlay Traffic in Heterogeneous Internet Topologies", "EMANICS: European Network of Excellence on Management of the Internet and Complex Services", and "Daidalos II: Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimized personal Services".

Fabio's main research interests are distributed systems, in particular peer-to-peer multimedia streaming supporting time shifting, network protocols, peer-to-peer storage, and incentives in peer-to-peer streaming systems. His doctoral thesis was supervised by Prof. Dr. Burkhard Stiller (University of Zurich, Switzerland) and Dr. Tobias Hoßfeld (Universität Würzburg, Germany).